INDIVIDUALIZED CARDIAC RESYNCHRONIZATION THERAPY:

NEXT GENERATION PACEMAKER CONTROLS

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Peter Jacob Hettwer

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Electrical and Computer Engineering

November 2015

Fargo, North Dakota

North Dakota State University
Graduate School

Title

INDIVIDUALIZED CARDIAC RESYNCHRONIZATION THERAPY: NEXT GENERATION PACEMAKER CONTROLS

By

Peter Jacob Hettwer

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Roger Green
Co-Chair

Dr. Dan Ewert
Co-Chair

Dr. Mark Jensen

Approved:

18 November 2015
Date

Dr. Scott Smith
Department Chair

ABSTRACT

Cardiac dyssynchrony (CD) causes some heart muscle regions to contract at different times, and current treatments do not help 30 – 50% of patients. In this thesis, multi-site pacing control schemes are created to quantitatively and automatically reduce the CD of ventricular wall accelerations by adjusting pacing times. Two and four left ventricular region models are investigated containing model variables that represent numerous muscle parameters. Optimization is performed using exhaustive search and genetic algorithm techniques, with particular attention paid to the latter with regard to development, parameter selection, and limitations. Relative to treatments firing all regions simultaneously, timing adjustment improves acceleration CD by up to 56%. Furthermore, simulations also demonstrate improvements to dyssynchronous region power generation and workload by up to 50% and up to 15% decrease in healthy region workload. Thus, the current model indicates it may be possible to improve acceleration CD by adjustments to regional firing times.

ACKNOWLEDGEMENTS

I would like to give special thanks to my advisor, Dr. Green, for his continued and consistent support and review of my work along with my family and close friends, without which I likely wouldn't have finished this endeavor. I would also like to extend thanks to my co-advisor, Dr. Ewert, for the conversations that sparked my research path, Dr. Jensen for his enthusiasm and willingness to be an external contributor, Sam Ogunyemi for his assistance in adapting his cardiac dyssynchrony model to allow interactions with my work, and Andrew Taylor for his assistance in paper editing.

DEDICATION

This work is dedicated to my family, close friends, and the love of my life who have been pillars of support through the challenges of life and graduate school.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

A.....Atrial

AF .....Atrial Fibrillation

AV.....Atrio-Ventricular

AP or AP_n.....Adjusted Parameter (for region n)

CD.....Cardiac Dyssynchrony

Cn.....Chromosome number n

CO or CO_{xx} .....Cardiac Output (for condition xx)

CRT.....Cardiac Resynchronization Therapy

EF or EF_{xx} .....Ejection Fraction (for condition xx)

EmaxD or $E_{max}D$ .....Maximum Elastance Dyssynchrony

EminD or $E_{min}D$ .....Minimum Elastance Dyssynchrony

ES.....Exhaustive Search

Fig .....Figure

GA.....Genetic Algorithm

Gen.....Generation

LA .....Left Atrium

LV .....Left Ventricle

MD .....Mass Dyssynchrony

MN .....Added Noise

N.....New

NA.....No Adjustment

N/A.....Not Applicable

ND.....No Dyssynchrony

NN.....No Noise

O.....Old

P_{genx} ..... Population for generation number x

RA.....Right Atrium

Ref.....Reference

RD.....Resistance Dyssynchrony

RV.....Right Ventricle

R_2D.....Region 2, All Dyssynchrony

$R_{234}D$.....Regions 2, 3, and 4, All Dyssynchrony

SA .....Sino-Atrial

SD .....Standard Deviations

$t_{xx,n}$ ..... Time for condition xx, region n

t_{Nxxn} ..... Normalized time for condition xx, region n

V.....Ventricular

%Δ.....Percent Change

LIST OF APPENDIX FIGURES

1. INTRODUCTION

1.1. Thesis Statement

Current CRT devices rely on patient/physician interactions in an attempt to set the proper pacemaker timing(s). An automatically tuned, multi-site pacing control device alleviates the need for manual, post-operation adjustment by a physician and should help to reduce the non-response rate in patients requiring CRT by adapting itself to the patient's individual needs. It is hypothesized that a multi-site pacing control scheme can be created via an optimization algorithm to reduce cardiac dyssynchrony (CD) by adjusting regional pacing times based on ventricular wall accelerations.

One optimization method investigated is exhaustive search. Exaustive searches create a grid of search points over the entirety of a search space and select the best solution based on some metric of performance. Another optimization method investigated is the genetic algorithm (GA). GAs are a stochastic method borrowing selection principles from biologic evolution to select and favor better performing solutions.

GAs think of problems as "black boxes" in that, they do not care how the problem works, but rather only concern themselves with the set of control knobs that can be adjusted, and a singular metric of how well those control knob positions "solve" the problem [22]. This unique feature allows them to find solutions to problems some other optimization methods cannot handle due to lack of objective function and search space continuity, linearity, derivatives, or other necessary features [8, 32].

Exhaustive search is a brute force search that tests and quantitatively evaluates all possible solutions within a specified search space to find the best one. In the case of problems with a continuous set of possible solution values, the search space is broken into a discrete subset

of values that can be tested and evaluated for the best solution within that subset. Exhaustive searching is generally used in cases when no efficient method is known to arrive at the best solution or as the standard by which to evaluate the ability of another search method to arrive at the best solution.

The intellectual merit of this research is that, presently, no pacemaker employs either a self-adjusting or multi-site pacing control algorithm for the treatment of CD. This research has the potential to advance cardiac resynchronization therapy (CRT). This work takes the first step in using ventricular wall acceleration from a CD model to develop a multi-site pacing control algorithm to quantitatively and automatically determine an optimal set of regional pacing times individualized to the model parameters at the time of control system initialization. This work does not investigate continuously adaptive techniques. A secondary goal of this thesis is to give an in depth tutorial of Genetic Algorithms, their creation, and their potential in solving a complex, biological system problem.

1.2. Introduction of Cardiac Function, Cardiac Dyssynchrony, Cardiac Pacing, and Cardiac Resynchronization Therapy

1.2.1. Cardiac Function

The heart is a complex electromechanical system in which muscle contraction is coordinated by an electrical stimulus from either the brain via the nervous system, or from one of the secondary firing nodes. (These nodes are known as the atrio-ventricular, AV, node and the sino-atrial, SA, node.) The structure and operation is summarized from [21]. Cardiac muscle is unique in that it is striated, like skeletal muscle, but involuntary, like smooth muscle. Like skeletal muscle, cardiac muscle is organized in a hierarchy with the muscle fiber being the base

cellular unit. Each fiber contains many parallel contractile structures known as myofibrils with each myofibril being further segmented into sarcomeres. Sarcomeres in turn, contain actin and myosin that interact and move following the sliding filament theory as described further in [21]. These structures can be observed in Figure 1.

The forces generated by these fibers, both passively, at rest, and actively, during contraction are dependent on a number of interactions including the number of parallel myofibrils, the velocity of contraction, and the initial length of the sarcomeres. During the contraction of a fiber under normal cardiac conditions, the fiber starts at an initial length with an initial tension known as the passive tension. Then, when electrically stimulated, the fiber begins to contract isometrically, generating increased tension without moving, and once the fiber generates enough force to overcome its load it contracts and shortens isotonically, with the same tension, while shortening in length. In the case of the entire heart, the load includes some portion of the force needed to overcome the pressure holding the chamber's valve closed, the fiber's mass, and a portion of the total friction and inertia present in the system. The muscle then isotonically lengthens returning to its resting length, and finally concludes by isometrically relaxing to its passive tension [21]. The length tension relationship is an important idea in attempting to understand the underlying concepts involved in cardiac operation and remodeling under non-ideal/non-normal conditions. Studies have shown that cardiac muscle length affects force generation by its influence on excitation contraction coupling [30]. Additional studies have shown that cardiac muscle stretch [31] causes hypertrophy of the muscle in the same manner that pressure overload (increasing cardiac work load) does [31].

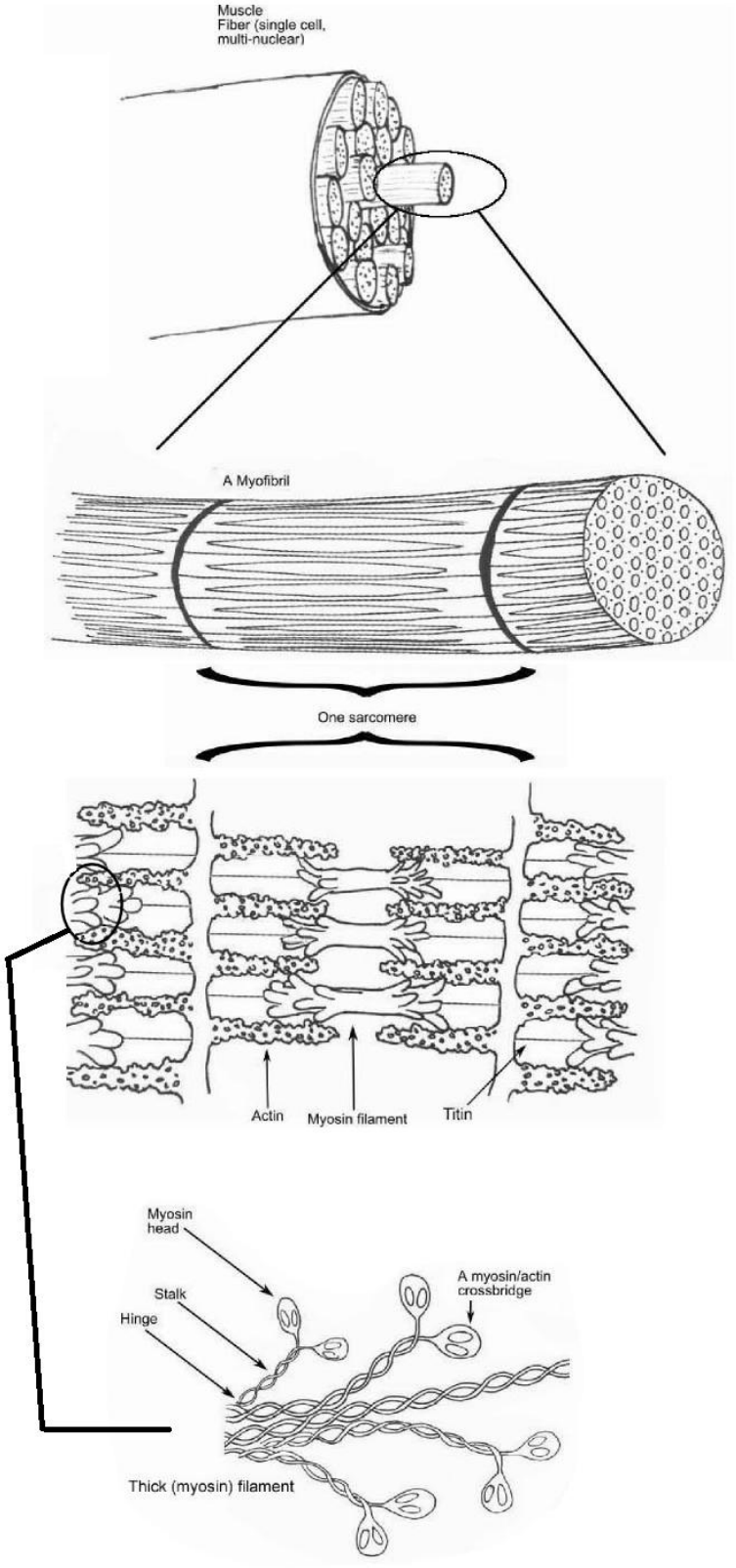

Figure 1: Figure of Muscle Structure. Modified from [35] under free use license.

1.2.2. Cardiac Dyssynchrony

During a normal cardiac cycle, cardiac tissue contracts in concert (together) as described previously, but cardiac dyssynchrony (CD) occurs when some muscle regions of the heart wall contract at different times causing early-activating regions of contraction to push blood volume into late-activating regions of contraction reducing ventricular wall accelerations [1, 2]. As a result, these changes in activation time cause decreased stroke volume and cardiac function [1, 2]. In addition, this abnormal stretching puts excess strain and stress on the late-activating muscle regions and changes myocardial blood flow that can lead to muscle tissue remodeling that is deleterious to cardiac function over time [1, 2, 3, 27]. Changes in ventricular wall thickness begin as early as 1 month after the start of pacing with early activated regions thinning and late activated regions thickening corresponding to the respective changes in work load [2].

1.2.3. Cardiac Pacing

Mobile and implantable cardiac pacing got its start in the 1950s and 1960s. The advent of the first battery operated, and totally implantable pacemakers with recorded long term correction of heart block by an implantable pacemaker occurred in 1960 [5]. In theory, the idea is relatively straight forward: the pacemaker (device used for cardiac pacing) assists in creating an electrical stimulus that forces the cardiac muscle to contract. Over the years, this device evolved from a single node pacing device with set timing and no sensing capabilities, to a single-atrial/bi-ventricular (3 node) pacing system with both sense and pace capabilities. The current system also has the ability to vary the heart rate via a sensor placed in the pacemaker box meant to detect body movement and approximate activity level [5]. But still, even with these advancements in pacemaker design, there still exist shortfalls with current pacing techniques.

At present, cardiac pacing is the only effective method for treating cardiac conduction disorders and sick sinus syndrome. Typically, pacing leads are anchored in the apex of the right ventricle; however, an increasing number of studies link RV apical pacing to detrimental cardiac remodeling and left ventricle (LV) function [6]. A number of studies are compiled in [6], and a few of these trials even associate RV pacing with eventual cardiac morbidity and mortality.

These detrimental changes are believed to result from the use of abnormal conduction pathways to achieve full cardiac contraction [6]. Both dual chamber, atrial (A) and ventricular (V), pacing and single chamber (A or V) pacing were associated with an increased risk of heart failure hospitalization of over 40%. A number of studies show long-term RV pacing may lead to ventricular dyssynchrony. Three studies presented indicate anywhere from 36% - 66% of patients exhibit LV dyssynchrony after RV apical pacing. In yet another study 26% of patients developed new-onset heart failure after 0.8 – 12.2 years of RV apical pacing [6]. Long term changes associated with right ventricle (RV) apical pacing include: changes in oxygen demand and regional blood profusion, asymmetric hypertrophy, ventricular dilation, decreased cardiac output, increased LV filling pressures, changes in myocardial strain, and inter/intra ventricular mechanical dyssynchrony among others [6, 26, 27].

1.2.4. Shortfalls of Current Pacing Techniques

While pacemakers are necessary to solve some cardiac problems in the short term, RV pacing techniques have been repeatedly shown to cause detrimental effects to overall cardiac remodeling and long term cardiac health [16]. For example, AV node ablation and permanent pacing is well established as a treatment for atrial fibrillation in cases that the patient does not respond well to pharmaceutical remedies. However, long term RV apical pacing, as is common in two lead pacing systems, has been shown to cause LV remodeling along with a decrease in left

ventricular ejection fraction and performance [16, 17, 26, 27]. A similar study investigated the effects of RV pacing after His bundle ablation and concluded that RV pacing adversely effected LV structure and performance in patients showing normal LV function and dimensions prior to pacing [29]. The bundle of His is a network of specialized cardiac cells that propagates electrical signals through the heart. Additional deleterious effects of RV pacing include left ventricular electrical and mechanical dyssynchrony, abnormalities in myocardial histopathology, latrogenically accentuated intra-ventricular conduction delay, congestive heart failure, myocardial perfusion defects and regional wall motion abnormalities, functional mitral regurgitation, increased risk of atrial fibrillation (AF) and heart failure in patients with sino-atrial (SA) node dysfunction, left atrial (LA) enlargement, promotion of ventricular arrhythmias, and activation of the sympathetic nervous system [17, 29].

The cause of the remodeling is believed to be related, at least in part, to the induced LV dyssynchrony caused by long term RV pacing [16, 27]. Studies have linked long term RV pacing to LV dyssynchrony in almost 50% of patients treated with AV node ablation for AF [16]. Another study tied LV remodeling directly to LV dyssynchrony by implanting pacemakers in otherwise healthy dogs and inducing LV dyssynchrony through RV pacing [27]. This result shows that variation in LV workload and LV remodeling is not solely the result of cardiac disease progression, but has at least some component tied to RV pacing induced LV dyssynchrony. However, this study conflicts with the findings of [28] that claims long term RV pacing alone appears unassociated with the development of heart failure, deterioration in ventricular function, or reduced survival in patients without an antinuclear antibody. The study goes on to claim that ventricular dyssynchrony can exacerbate the progression of heart failure in patients with compromised “cardiac reserves”, but RV pacing’s effect on cardiac function in

patients without structural heart disease is still not fully defined [28]. Regardless of these conflicting studies, the fact remains that there is a need for advancements in cardiac pacing technology for at least a subset of current pacing eligible patients.

Current pacemakers utilize preset timings to determine pacing control, and use at most two ventricular pacing sites. These methods rely on doctor/patient interactions to set relative firing times for the individual nodes and do little to objectively minimize the amount of effort required by the doctor to fine tune pacemaker operation for individual patients. A never attempted approach is to use measured ventricular wall accelerations to create a pacing sequence unique to every patient with the goal of reducing CD without a doctor performing the tedious operation of hand tuning each patient's device.

1.2.5. Cardiac Resynchronization Therapy

As of 2011, heart failure affected 5.8 million patients in the US, with an addition of about 500,000 annually, and remains a major cause of hospitalization and death [25]. It is believed that up to 20-30% of all congestive heart failure patients are also afflicted by ventricular dyssynchrony [17]. Cardiac Resynchronization Therapy (CRT) in the form of biventricular or left ventricular pacing has been acclaimed as a new mode of non-pharmaceutical, non-surgical (non-transplant) therapy for patients with moderate to advanced heart failure [17]. In current Cardiac Resynchronization Therapy (CRT), pacemakers compensate for this difference in activation times by pacing two ventricular sites in an attempt to get all regions of the ventricles to contract in concert [4]. However, 30-50% of patients receiving CRT do not have their dyssynchrony reduced. One reason for this may be scarring of tissues that create islands of viable tissue that may not contract in synchrony by using only two ventricular pacing sites [4]. The introduction of biventricular pacing in current CRT devices has helped achieve mechanical

cardiac synchrony in patients suffering from ventricular dyssynchrony improving quality of life and reducing hospitalization rates for many patients [25, 26]. CRT has been shown to reverse the detrimental remodeling of the LV for responding patients by reducing LV volume, and increasing LV ejection fraction [25, 26]. In addition to this, one study calculates that for every nine devices implanted, one death and three hospitalizations are prevented [26]. However, even with these advances in cardiac pacing techniques, only about one in three heart failure patients meet the requirements for current CRT methods [25].

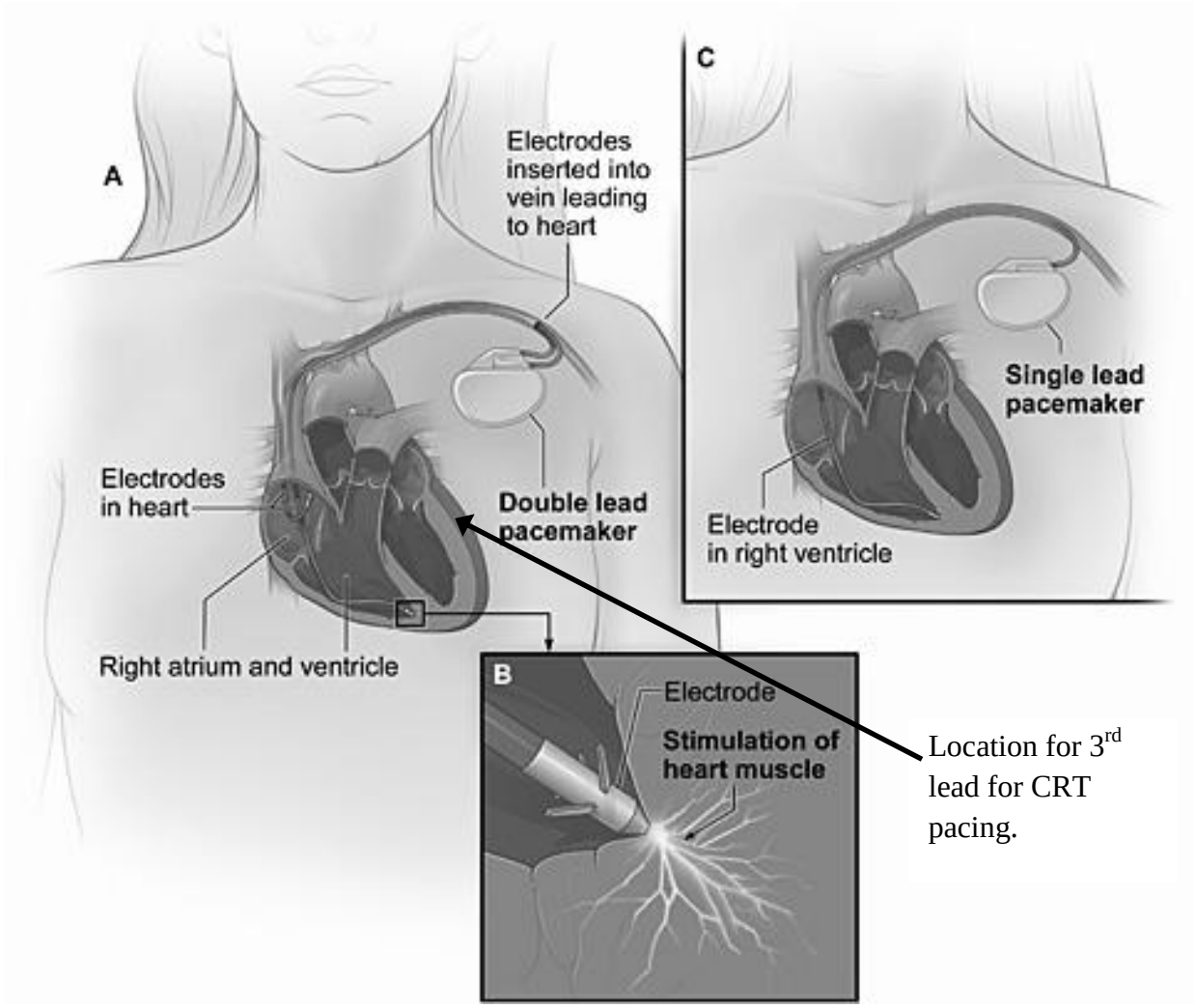

Figure 2: NIH Diagram of 1 and 2 lead pacemaker lead placement [34]. 2nd ventricular lead for CRT pacing would be located on the LV free wall. Arrow added to show location of 3rd lead for CRT pacing.

1.3. Introduction of Genetic Algorithms

1.3.1. History of Genetic Algorithms

Genetic algorithms were initially developed in 1958 by Bremermann [20] but popularized by Holland and his students and formally applied the mechanisms to computer science [20, 22]. This base model developed by Holland and associates is referred to as the canonical genetic algorithm (sometimes also referred to as the simple genetic algorithm) [22, 32]. This led to various advancements including variable length chromosomes by Kotani, Ochi, Ozawa, and Akazawa [20], and Bremermann's further advancements, by being the first to implement a real-coded Genetic Algorithm [20] with the idea that future computers could implement his more advanced concepts and methods [20]. Additionally, many other variations have been, and can be made on the canonical genetic algorithm to better tailor the algorithm to a specific problem of interest.

1.3.2. Introduction of Genetic Algorithms

A genetic algorithm (GA) is an optimization algorithm branch of evolutionary computation that imitates the biological processes of natural selection through reproduction to find the "fittest" solution [8, 13, 32]. GAs are global, stochastic search algorithms that operate on populations of current "approximations" that begin as a random set of test parameters, and as the search proceeds, the population is modified by exploiting characteristics from favorable solutions previously tested [32]. By operating on populations of potential solutions, GAs lend themselves readily to parallel computing techniques [32]. These algorithms are more powerful than either random search or exhaustive search algorithms in that they converge to their solution much more consistently and quickly on average respectively than either of the other two search

types, and yet require no extra information on the problem such as solution space derivatives or gradients [8, 13].

GAs think of problems as "black boxes" in that, they do not care how the problem works, but rather only concern themselves with the set of control knobs that can be adjusted, and a singular metric of how well those control knob positions "solve" the problem [22]. This unique feature allows them to find solutions to problems some other optimization methods cannot handle due to lack of continuity, linearity, derivatives, or other necessary features in the objective function and/or search space [8, 32]. For example, GAs provide a strong framework for solving nonlinear, multi-objective, multi-modal, and other complex system problems [13, 22]. One of the few requirements is that the parameters being optimized must be able to be represented by encoded strings, such as binary strings [22, 32]. In addition, since GAs act on the coded variable rather than the variable itself, it is suited for the use of structure objects as well [13, 32]. GAs have the potential to converge to nearly global optimal solutions given that the GA is set up correctly for the problem since it is possible for them to search the entire solution space [13, 32].

The downfall for some optimization algorithms that rely on derivatives and gradients, like gradient descent, is that they target and move toward the closest minimum or maximum from the starting point. Algorithms like gradient descent work extremely quickly and efficiently for problems where the solution space is monotonically increasing or decreasing. However, to come to this conclusion, one needs previous knowledge of the search space being investigated. GAs have the inherent benefit of also working for non-monotonic, discontinuous, and noisy functions since they work on a population of possible solutions each generation and no single solution inherently impacts all following solutions [14, 32].

Another benefit of GAs is that because of the stochastic population based search, they have the possibility to find a family of similarly fit solutions just by re-running the algorithm [14]. GAs can be applied in two main areas of control engineering: off-line design and on-line adaptation, and are discussed to some detail in [32]. Off-line optimization is useful in operations that do not require constant updating to "fine tune" a system after it is set and turned on. In on-line applications however, one must be cautious of the direct application of "weak" individuals within the population operating directly on the system of interest since the consequences could be severe. In order to operate safely in on-line applications, it is common for the GA to operate on a fairly accurate model of the system and only indirectly tune the controlled system. The other option, though arguably less safe, is allowing on-line operation on the system if it is known the system is sufficiently fault tolerant and robust to tolerate the level of exploration utilized and required by the GA. In addition, on-line applications usually require faster rates of convergence, usually at the cost of decreased robustness [32]. One example of decreased robustness could be having a wider convergence window around the optimal solution.

When used carefully, GAs can be used to create adaptive systems that can tolerate changes in the system being controlled [32]. Almost any implementation of a GA has a few key components that are explained in detail in section 2.1. But, in short, all GAs use some form of the following components: an optimization function, a population of chromosomes, a chromosome selection scheme (choosing mating chromosome pairs), a method of chromosome crossover (how data are traded between paired chromosomes), and a method for random chromosome mutation.

1.3.3. Operation of a Basic Genetic Algorithm

Most GAs follow a similar set of steps to systematically and stochastically test potential solutions within the global solution space of the problem. After determining specific operational values for the GA, which are usually tuned for each individual problem, GAs follow much the same set of operations: mate selection, crossover, mutation, objective function score computation, etc. The GA for a specific problem may make slight variations to this set in order to implement or remove certain operations, such as elitism, chromosome encoding variations, or others as desired by the designer to elicit specific responses desired for the system being optimized. Specific values have to be selected for a number of variables that have to be considered carefully to implement a GA successfully for each specific problem including: defining the objective function, selecting an encoding scheme for the possible solutions, selecting population size, determining crossover criteria/rates, selecting mutation rates, along with multiple others. A high level, operational overview can be helpful in visualizing the various considerations that must be weighed when using GAs. Figure 3 gives a graphical representation of the general steps involved. First, the initial population of chromosomes is randomly generated from the defined solution space [14, 15, 20, 32]. These chromosomes are then evaluated by computing a fitness value using a specific optimization function that determines how well each individual chromosome satisfies a specific set of criteria unique to the problem being solved [14, 15, 20, 32]. Next, the results of this testing are utilized to determine parent selection (pairing of chromosomes) by one of many pairing protocols such as rank selection, roulette wheel selection, and others that are discussed in more detail later in chapter 2 [14, 15, 20, 32]. The paired chromosomes then undergo a crossover operation by which the two chromosomes trade some portion of their information with each other [14, 15, 20, 32]. These new chromosomes undergo a

second operation, mutation, by which information within the chromosomes can be randomly changed outside of trading material with another chromosome [14, 15, 20, 32]. The chromosomes now having undergone crossover and mutation constitute the population of a new generation, and the cycle repeats from the step of testing the individual chromosomes against the problem's optimization function [14, 15, 20, 32]. GAs do not explicitly remember fitness results from previous generations, but due to the selection processes employed to generate the next set of test solutions, the best solutions from the current generation have the best chance to be represented in the subsequent generations [18, 32]. Additionally, certain other techniques such as elitism can be utilized to ensure the best solution(s) from the current generation is automatically passed to the subsequent generation. These are also discussed in further detail in subsequent sections in chapter 2, but most notably in section 2.1.

GA papers generally use a combination of biological and traditional optimization terms to explain the methods of how a GA goes about finding a solution to its specific problem. Definitions of biological GA nomenclature are discussed in section 2.1.

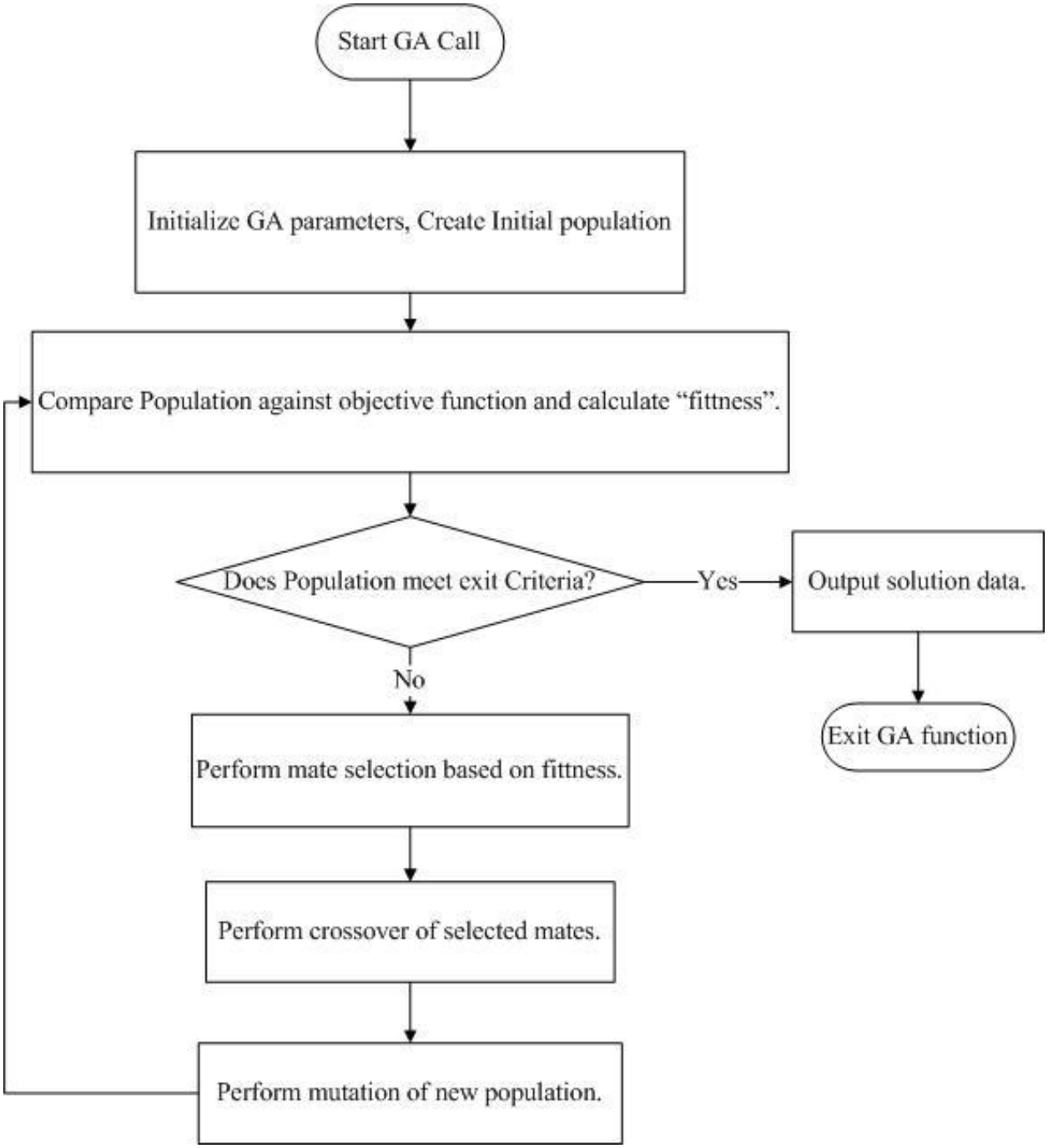

Figure 3: Generic GA Flowchart

2. GENETIC ALGORITHMS AS THE NEXT STEP IN PACEMAKER CONTROL SYSTEMS

This chapter outlines GA criteria that need to be considered in creating a GA based optimization algorithm for a specific problem culminating in a design for a GA that indicates when pacing should occur during a normalized beat. This pacing is timed to reduce cardiac dyssynchrony by reducing regional differences in ventricular wall accelerations. Sections 2.1 and 2.2 outline the components necessary to create a GA.

The first step necessary to create a GA for the purpose of finding regional cardiac firing times is to cast, or map, the problem onto a GA architecture. Specific GA properties (such as elitism) can aid in ensuring operations act in the most beneficial way for our problem. The second step is to test the problem with a MATLAB coded GA (from step 1) and a simple, MATLAB coded model, referred to as the walking sinusoid model, with the goal of testing the GA itself to ensure the coded algorithm operates true to GA nature and verify the algorithm before attempting to work with more complex. Finally, after the GA is in an acceptable state with the sinusoid model, it is tested on a more complex model of CD. This testing is designed to obtain preliminary data on the feasibility using a GA to reduce CD by measure of ventricular wall accelerations. These results are compared against the discrete exhaustive search of a sufficient subset of the GA solution space to quantify how well the GA did in two categories: A.) reducing ventricular wall acceleration CD and B.) comparing the time required to obtain the GA solution to the time required to perform the exhaustive search of a similar search space.

2.1. GA Terms, Operators, and Options

Figure 4 shows a basic overview of all GA terms and operators in one graphical representation for easy reference. Individual terms, operators, and options are detailed further in

the following sub-sections as a tutorial for basic GA utilization and implementation. The tutorial culminates in a selection of specific criteria for mapping the main thesis objective of determining if specific timings can reduce ventricular acceleration CD.

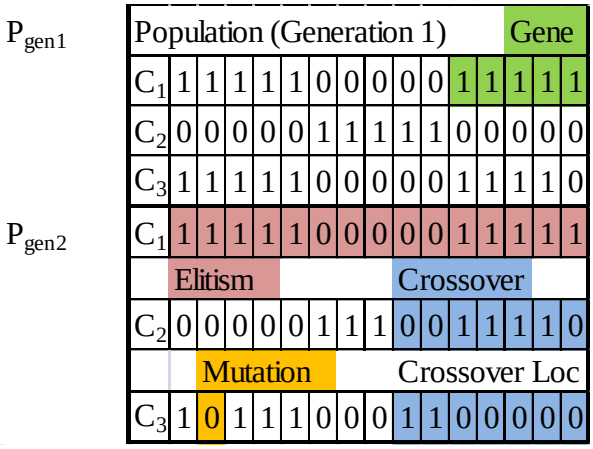

Figure 4: Generic GA Representation of Population Components and Operations

2.1.1. Chromosomes

A chromosome, C_n, is single possible solution where n is the solution's number comprised of a set of coded variables used by the GA and the optimization function. The optimization function uses the variable set to determine the relative "goodness" of the solution in comparison to all other solutions of that solution set, and the GA uses the coded variable string to generate subsequent generations.

2.1.2. Genes

A gene is a specific variable within the chromosome that comprises part of a single possible solution.

2.1.3. Populations

A population is the entire set of chromosomes that exist at any one time. The size of the population is decided by the user. Selection criteria are detailed in section 2.2.1.

2.1.4. Generations

A generation is the population of all chromosomes that are tested using the optimization function to receive the fitness for each chromosome to determine the ranking profile necessary to generate individual chromosome selection criteria and percentages prior to generating a new population based on the relative rankings assigned by the optimization function. The selection criteria can be one of a number of different operations, each of which are detailed in later sections.

2.1.5. Mating Operations

Crossover paring, paring, or mating is the operation by which the relative rankings assigned to the chromosomes of a generation are used to select which chromosomes "mate" and exchange information with each other. These parings can be created by a number of different selection operations such as roulette wheel, tournament selection, and rank selection.

2.1.5.1. Roulette Wheel

In roulette wheel (relative probability) selection, selection probabilities of the previous generation for mating are based off how well a given solution is evaluated by the objective function [9, 14].

For a maximization problem, individual fitness is divided by the sum of all individual fitness values. Where 'cost' is the vector of values returned by the optimization equation for the generation, cost(n) is the cost value for a chromosome n, the probability of selection, P_{n_max}, can be determined as follows:

$$P_{n\_max} = \frac{cost(n)}{\sum_{i=1}^{N} cost(n)}$$

For a minimization problem with the same conditions, probability, P_{n_min}, may be computed by an equation such as:

$$P_{n\_min} = \frac{max(cost) - cost(n)}{\sum_{i=1}^{N} cost(n)}$$

It is worth noting that using this equation, the worst fitting case has no chance of reproduction.

This type of selection can be viewed as having a few problems in that selective pressure can be quite high in the first generations if one of the chromosomes dominates fitness with respect to the others, and as the search continues with the population converging, selective pressure can decrease substantially [10]. Figures 5 and 6 give a visual representation of relative fitness selection.

P_{genX}

																F	F_{rel}
C_1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	15	0.395
C_2	1	1	1	1	1	0	0	0	0	0	1	1	1	1	0	12	0.316
C_3	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	9	0.237
C_4	0	0	0	0	0	1	1	1	1	1	0	0	0	1	1	2	0.053
C_5	1	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
															Total	38	

Figure 5: GA Roulette Wheel Selection Criteria

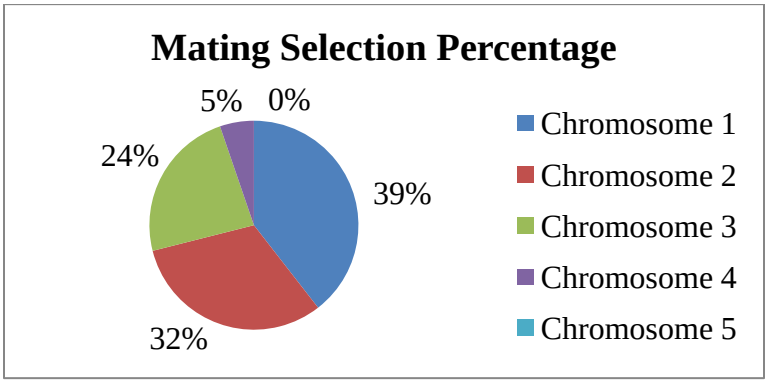

Figure 6: Roulette Wheel for Relative Fitness Selection of Mating Probabilities

2.1.5.2. Tournament Selection

In tournament selection, two individuals are randomly selected from the previous generation and the fitness values compared; the more fit of the two is selected as one of the mates to create the next generation [10, 14]. Figure 7 gives a visual representation of tournament selection.

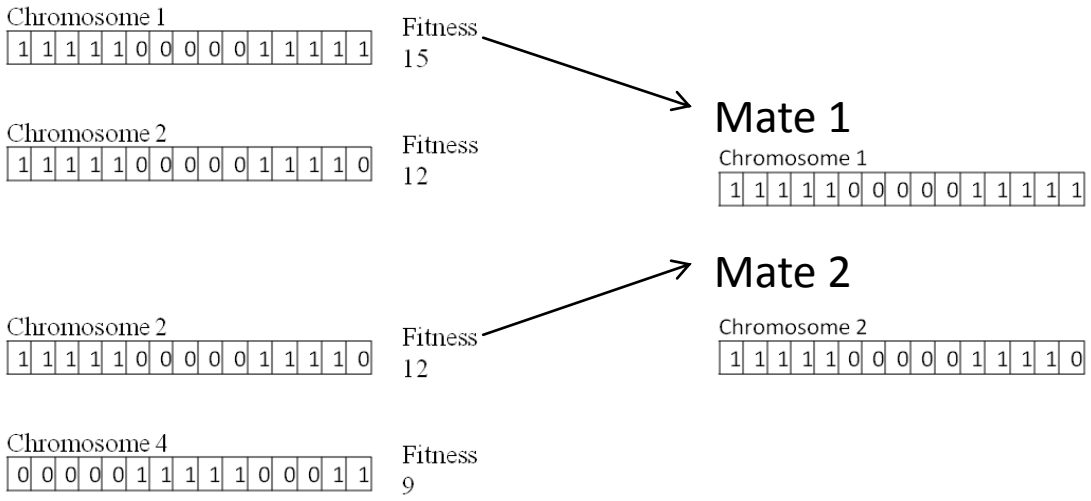

Figure 7: GA Tournament Selection

2.1.5.3. Rank Selection

Rank selection is a more controlled roulette wheel in that the probabilities of selection do not rely on the relative fitness of a solution with respect to its competition in the generation [14]. In other words, the probabilities of selection for a given rank are fixed between generations, and the chromosomes are given one of the probabilities based off their fitness with respect to other chromosomes in the generation. This type of selection results in a selective pressure that is more consistent and controlled when compared to that of a traditional roulette wheel [10, 14]. The values for individual selection probabilities can be individually selected and assigned by the GA

designer or computed by a weighting function of the user's choice. Figures 8 and 9 give a visual representation of rank selection with selection probability arbitrarily assigned for demonstration.

P_{genX}

																F	F_{rel}	P_{sel}
C_1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	15	1	0.40
C_2	1	1	1	1	1	0	0	0	0	0	1	1	1	1	0	12	2	0.25
C_3	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	9	3	0.20
C_4	0	0	0	0	0	1	1	1	1	1	0	0	0	1	1	2	4	0.10
C_5	1	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	5	0.05
															Total	38		

Figure 8: GA Rank Selection Criteria

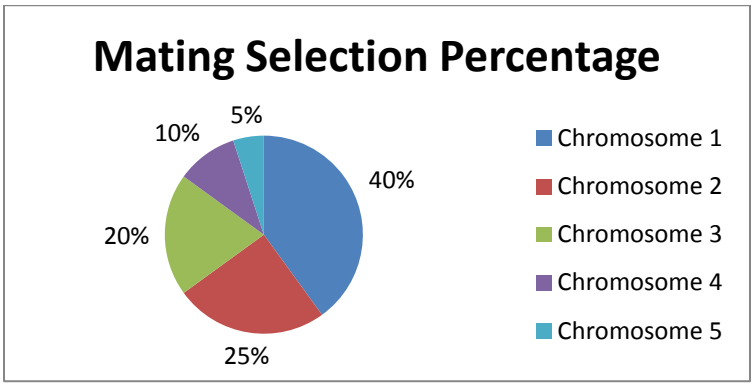

Figure 9: Roulette Wheel for Rank Selection of Mating Probabilities

2.1.6. Additional Selection Criteria

In addition to the mating selection methods discussed previously, there are options to add specialized functionality to a GA for specific problems should the user decide to implement them. Two such operations, elitism and steady-state selection are explained in further detail.

2.1.6.1. Elitism

Elitism is the GA equivalent of survival of the fittest. The best chromosome(s) from the previous generation are ensured to continue to the next generation without regard to crossover or

mutation mechanics [9]. The remaining selections are performed in one of the other ways discussed previously in section 2.1.5.

2.1.6.2. Steady-State Selection

In steady-state selection, the worst chromosomes are discarded before mating selection takes place and only the best chromosomes are considered for mating selection. This can be considered inverse elitism. If implemented, the user can select the number of discarded individuals, or discard individuals with less than a specific level of fitness with a threshold on the maximum number of individuals removed. The discarded chromosomes are replaced via crossover mating where mates are selected by one of the methods discussed previously in section 2.1.5. The remaining chromosomes continue without replacement [9].

2.1.6.3. Generation Gap

Generation gap is the percentage of a generation that is attempted to be replaced from one generation to the next [12]. In any case that the generation gap is not 1.0 (100%), the remaining positions in the next generation are filled with individuals from the current generation selected randomly by a uniform distribution [12].

2.1.7. Crossover Operations

Crossover is the operation by which two chromosomes of the previous generation trade information. This operation can be performed by a number of different schemes like single point, multiple point, and uniform crossover, all discussed in further detail. The following examples for each of these operations use binary encoded chromosomes, though the operations work with other encoding schemes as well. Encoding schemes are discussed in more detail in section 2.1.9.

2.1.7.1. Single Point Crossover

Single point crossover selects one point in the chromosome after which, all information between the mated chromosomes is exchanged. The crossover location is generated randomly and can occur anywhere along the length of the chromosome. Figure 10 shows a representation of single point crossover between two chromosomes in sequential generations.

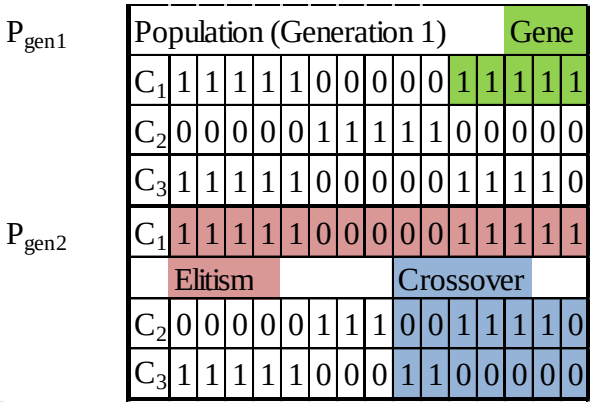

Figure 10: GA Single Point Crossover

2.1.7.2. Multiple Point Crossover

Multiple point crossover selects two or more points between which information is exchanged between the mated chromosomes. The crossover location is generated randomly and can occur anywhere along the length of the chromosome. Figure 11 shows a representation of multiple point crossover between two chromosomes in sequential generations.

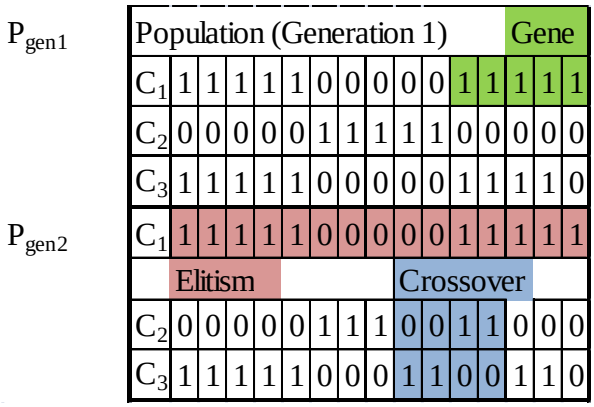

Figure 11: GA Multiple Point Crossover

2.1.7.3. Uniform Crossover

In uniform crossover, information is randomly exchanged between the mated chromosomes [9]. The number of crossover points can be set, or random in addition to the locations of crossover being generated randomly. Figure 12 shows a representation of uniform crossover between two chromosomes in sequential generations.

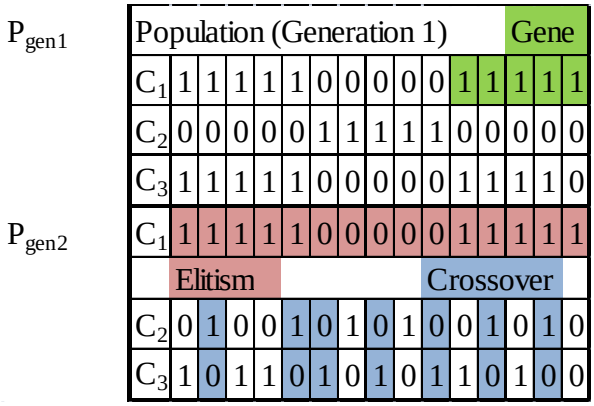

Figure 12: GA Uniform Crossover

2.1.8. Mutation Operations

Mutation is the operation by which a chromosome can be randomly changed outside of crossover to introduce new variation in the population. Depending on the encoding scheme, mutation can be implemented in multiple ways. Only binary encoded and value encoded

mutation are discussed in more detail here. The others are specialized in their applications and outside the scope of a general GA tutorial and the resulting implementation used in this thesis to reduce ventricular wall acceleration CD by regional timing modifications.

2.1.8.1. Binary Encoded Mutations

In binary encoded mutations, a random number of bits based on a pre-selected probability are inverted randomly within a generation of chromosomes [9]. Choice of this pre-selected mutation probability is addressed in section 2.2.2. Figure 13 shows a representation of binary encoded mutation of a chromosome between sequential generations.

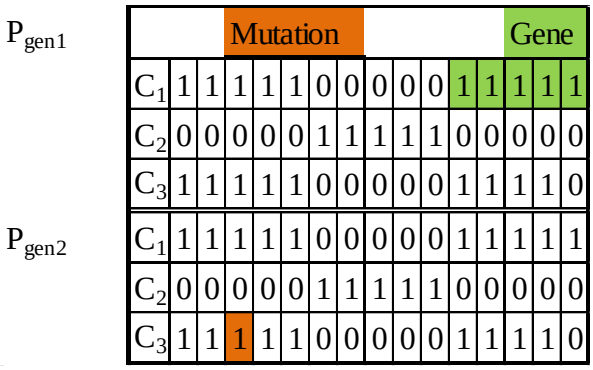

Figure 13: GA Binary Encoded Mutation

2.1.8.2. Value Encoded Mutations

In value encoded mutations, a random number of values based on a pre-selected probability are changed by adding or subtracting a small number [9]. Depending on the programmer's preference, the 'small number' could possibly be fixed, or be a random distribution of the programmer's choosing around a mean value of interest, or within a specified range. Choice of this variation value depends largely on the GA designer's understanding of the problem attempting to be optimized.

2.1.9. GA Encoding Scheme

The encoding scheme is the method by which individual variables are stored and manipulated by the GA. These encoded values are then transformed when needed for use by the objective function. Encoding schemes can vary between GA implementations. Some common implementations are binary, and value encodings. These two types are discussed in more detail in following sections. Other encodings are possible, but are specialized in nature and outside the scope of the desired generic GA tutorial and the resulting GA implementation used to reduce ventricular wall acceleration CD by regional timing modifications.

As for which encoding is better (binary or value), results are unclear. The results depend largely on the problem being solved, and other factors such as the resolution (number of bits) representing the genes [10]. For the problem at hand, binary encoding was selected to increase computation efficiency if ever ported to an embedded processor device and to reduce dependence on user selected parameters as compared to value encoded GA implementations in an attempt to make a more universal GA optimization algorithm.

2.1.9.1. Binary Encoded GAs

In a binary encoding scheme, all genes within the chromosomes are broken down and represented by a single binary string [9]. This is the most common GA encoding scheme since binary operations lend themselves so readily to necessary GA operations [9]. Since chromosomes are treated as a single binary string, gene start/end locations are tracked by the coding of the GA implementation. The way values can be encoded in binary numbers can vary, and each has its own merits and downfalls [10].

2.1.9.1.1. Binary Coded Decimal (BCD)

A standard BCD uses the standard binary counting scheme to keep track of successive values. Where N is the number of bits and the values for the binary digits (b_n) are 0 or 1, the decimal value of the number can be evaluated as:

$$\sum_{n=0}^{N-1} b_n 2^n$$

The ease of BCD conversion to decimal allows for efficient use in optimization functions designed using traditional mathematics and computer architecture.

2.1.9.1.2. Binary Grey Codes

A binary grey code represents decimal values ('real-numbers') as binary strings where consecutive decimal values vary by only one bit in the binary code [10]. Standard encoding tables can be used for this representation if the number of bits required is known, or can be custom created for a specific task and/or bit length. This type of encoding can help mitigate drastic value changes if a mutation were to occur at what would otherwise be a significant bit location in a gene. However, since this encoding is not easily switched between bit lengths, the number of bits required for this problem is not well defined, and implementing such an architecture would lose a great deal of the flexibility desired, it is not used for the problem at hand.

2.1.9.2. Value Encoded GAs

In a value encoding schemes, each chromosome is represented by a string of real values for genes including: integers, decimals, characters, complicated objects, and more [9]. This type of encoding is good for some very specialized problems, but may require special considerations to be made in implementing problem specific crossover and mutation functions [9]. More recent

evolutionary programming communities advocate real number encodings [10]. One reason for this may be in helping mitigate how drastic a change can be made by mutation to a critical value.

2.1.10. Optimization Function

An optimization function is the function by which the GA ranks the relative fitness of a specific chromosome. The optimization function is largely what specializes a GA to perform its specific search. This function can be anything from a simple difference of values or sum of squared error equation, to a complex, specialized function meant to exploit known aspects of a problem's search. Although prior search space knowledge is helpful, a GA does not require this prior knowledge to perform its search lending to the power of GA implementations. GAs do not explicitly remember the fitness of previous solutions from one generation to the next, rather the fitness is utilized to effect mating selection frequencies, and propagate more fit individuals in higher frequently than less fit individuals [18].

2.1.11. Stop Conditions

A stop condition designates a point that, once reached, causes the GA to quit searching for a more optimal solution, and exit its search returning the best solution found. Some examples of stop criteria include terminating the GA after a certain number of generations are tested, after the evaluation of the objective function reaches a specific value, or if no improvements have been observed after a specified number of generations [14]. In the case of the GA implemented for this thesis, all three stop conditions are used. Use of a specified objective function evaluation value allows for the case of exiting early if the ideal solution is found. Exiting after a specified number of generations allows for a fail-safe where the GA would terminate in the case where the population did not converge quickly enough. Finally, exiting after improvements have not been observed in a user determined number of generations allows for exit under the assumption that

the population has converged to a value and further improvement is unlikely except in the case of mutation. Under purely stochastic conditions, relying on only mutation to find solution improvements limit its usefulness after the population has converged and crossover can no longer aid in solution improvement.

2.2. GA Parameters

In addition to the basic GA operations discussed in section 2.1, there are a number of parameters that depend on user selected values and can drastically change implemented GA performance by orders of magnitude, even within a specific optimization problem being considered [11]. While there are guidelines for the ranges of specific values, care must be taken in setting values for a specific problem to balance minimizing the run time and maximizing the effectiveness of the search results. Investigations in the effects of population size and mutation rate on GA performance are made in this thesis, outlined in section 5.2.6 and discussed in section 6.2.5.

2.2.1. Population Size

Population size is the number of chromosomes in each generation of a GA search. Arguably, population sizing is one of the most important parameters set for successful GA operation on a specific problem [23, 24]. Too small a population can cause premature convergence, and as a result, a poor solution, while too large a population results in wasted computation time due to overly slow convergence to the optimal or near-optimal solution [23, 24]. Population size has been agreed to depend directly on the "difficulty" of the problem [23]. This means that the more "difficult" or the larger the number of adjustable parameters for the problem is, the larger the population size must be to accommodate the increased level of difficulty and prevent premature convergence. Initial trials of the implemented GA were run

using a fixed population size to isolate the number of possible factors being changed between test cases. Population size changes are investigated for a subset of the cardiac dyssynchrony model test cases and are discussed in section 6.2.5.

Multiple investigations reviewed in [11] speculate the best population size to be 20 - 30 chromosomes. Another study in [11] advocates for 'small' populations but does not elaborate on specific size, while [14] states population sizes of 20 to 500 are common, but also says there are few, if any, rules available to help determine the correct size. A third paper summarizes that even after years of research, population size is selected by little more than trial and error for any particular problem instance, and further goes to claim that the larger the initial population, the greater the chance that the initial population contains a chromosome representing the optimal solution, or an optimal gene [19]. While increasing population size can improve solution accuracy, it also has the less desirable characteristic of increasing the number of generations required to converge an entire population to a small number (1 – 2) of possible solutions [19, 24].

There is a trade-off between number of generations required for convergence to the correct solution and population size. Generally speaking, increasing the size of one decreases the other, but additional caveats can be present in deciding which is better. Repeated shorter runs with smaller populations can be better for finding multiple solutions that are acceptable, while longer runs with larger populations can be better for finding a single "best" solution [14]. Longer runs with a higher population sizes can end in better solutions. However, since the GA is stochastic in nature, it is still considered better in most cases to run multiple trials and average the results from each [14].

There are a number of ways to optimize population size. Two ways discussed in [23] are self adaptation of the GA either prior to executing a GA run (causing all generations in the GA to have the same population size), or self-adaptation during the GA run (allowing varying population sizes during the GA run) based on computed fitness values. Theoretically, population size can be coded to vary on any number of parameters the GA designer wanted as long as it aids in allowing the GA to solve the desired problem. These automated methods can be hard to implement since this largely depends on prior knowledge of the problem to guide the self-selection criteria. Thus, most GA implementations and implementers use trial and error to size a population that is appropriate for their problem of interest [19, 23, 24]. The results of [14] indicate a population size of 50 was ideal for their test case. Results for another problem with 3 control parameters operated on by a canonical genetic algorithm indicated a population size of 100 as ideal [24]; however, this can easily vary depending on values selected for other GA parameters so trial and error may still be the most reliable and simplest way to select a population size for a specific problem of interest.

2.2.2. Mutation Rate

Mutation rate is the probability that a random change occurs within a chromosome. In the event of binary encoded chromosomes, it is generally implemented as the probability that an individual bit inverts outside of crossover (changes from 0 → 1 or 1→ 0). A couple in-depth investigations discussed in [11] place the optimal value for mutation rate between 0.005 and 0.01 (0.5% and 1%) and is agreed on in [14] while another study investigated places it between 0.05 and 0.35 (5% and 35%) [11]. A third paper claims ideal mutation rates are between 0.001 and 0.01 (0.1% and 1%) [15].

Another study investigated combinations of changes to population size and mutation rate and showed better performance in populations sizes of 4 – 8 with a mutation rate of 15% than in population sizes of 64 – 128 with a mutation rate of 2% [11]. [11] goes on to conclude that for the problem they investigated, optimal population size should be less than 16 and mutation rate be between 5% and 20%. While, mutation rate values may vary greatly depending on the study referenced, the problem being investigated, and the combined interaction between mutation rate and population size, it is worth taking into consideration when investigating possible ways to streamline a more generic GA to a custom tailored one specific to the designer's problem.

2.2.3. Crossover Rate

Crossover rate is the probability that two paired (mated) chromosomes exchange information. In the event of crossover in binary encoded chromosomes, a second value is generated randomly to determine at which point the chromosomes' information is exchanged. In the event that crossover does not occur, both chromosomes are copied to the next generation without exchanging information. Desirable GA performance was shown to rely on a 'high' crossover rate [7]. Another paper further clarifies typical ranges for crossover rate as generally being between 0.6 and 0.95 (60% to 95%) [15]. Reducing crossover rate effectively reduces the number of new possibilities created in the subsequent generation [12]. Crossover rates of 0.6 (60%) and above show favorable results in GA convergence to an optimal solution while crossover rates being too high (1.0 or 100%) show the same detrimental effects of being too low (<0.4 or 40%) [12]. The theory is if crossover rate is too high (100%) favorable chromosomes to disappear more quickly than new favorable chromosomes can be created while the rate being too low (<40%) causes less exploration than is necessary to find new, favorable solutions [12]. In our case, we used a crossover rate of 100% since we implemented elitism to ensure the best

chromosomes do not disappear and wanted to promote as much exploration as possible with the remaining individuals in each generation.

2.3. GA Parameters in Context of Cardiac Dyssynchrony: Mapping the Problem

From previous discussion, it is apparent that there are many ways to create an algorithm that uses the required basic principles of GAs to perform the task of sifting out a near optimal solution from an initial pool of randomly generated possibilities. A single combination of the possible components previously discussed must be selected to create a starting point from which to fine tune a GA to specifically solve the problem of ventricular wall acceleration CD. Decisions on GA parameter values could have a direct impact on a future implantable design. Thus, the goal is to select parameter values well in the initial design phase to reduce the redesign and retest time between model based implementations, and those solutions destined for implantation. From an end goal point of view, faster convergence and ensuring that the best solution is never lost is of utmost importance considering the goal is to start helping the patient as quickly as possible. Additional trials investigate the effects of variations in population size and results are presented in section 5.2.6 and discussed in section 6.2.5.

All things considered in terms of population size, a population size of 20 was selected following the recommendations of [11] for the majority of GA test cases.

The mutation rate for the majority of GA test cases is selected as 0.001 (0.1%) since it is the most agreed upon value in literature with the continued reasoning that too high a mutation rate would increase time to convergence, yet having no mutation would allow no chance for the GA to explore outside the region created by the initial random population. Additional trials investigate the effects of variations in mutation rate and results are presented in section 5.2.6 and discussed in section 6.2.5.

Again, if implemented in hardware, the outcome of the GA will directly impact a person's health. It is imperative that the best found solution is never lost between generations due to crossover replacement. With this in mind, elitism is implemented to preserve the best two solutions from one generation to the next. Two is chosen because population sizes are generally even to facilitate efficient mating and crossover implementations with the fixed arrays and matrices that are present in the coded GA structure. An even number of elite members keeps the remaining population even as well. In addition, if elitism is ever removed from the GA architecture, no modifications to other operations are necessary.

The mate selection probability is determined by a relative probability roulette wheel with the idea that variable mating probabilities allow for more rapid population convergence which is desirable in a real time design as previously discussed.

With the implementation of an elitism mechanism, crossover rate is selected as 1.0 (100%) since the best solutions are always conserved with elitism, and maximum exploration is desired from the remaining population for more rapid convergence.

Since determining a "good" set of stop conditions is one of the more tricky sets of values to choose correctly, the initial number of generations was set at 50 for the CD model with the idea that it would create a "hard" stop condition to avoid getting stuck in a loop finding only marginally better solutions. Exit criteria were also created to exit the GA if the objective function evaluated to "0", or a perfect fit, and if no improvement was observed between populations for 5 consecutive generations which indicates the population likely converged to a single value and only a lucky mutation would have a chance at improving the solution.

The minimization metric (optimization function) that is used for these trials is a standard sum squared error equation:

$$Cost = \sum_{k=0}^{K} (ideal(k) - \ experimental(k))^2$$

where cost is the value being minimized, K is the inverse of the sample rate, or the number of computed points per period, ideal is the waveform being used as the control, and experimental is the waveform being modified by the optimization algorithm.

Parameter encoding is chosen as standard binary coded decimals (BCD) due to its computational efficiency and flexibility in implementing multiple gene bit lengths. Values encoded are represented as fixed point numbers between 0 and 1 and scaled to the correct range(s) for use in each the CD and sinusoid models. Other possibilities for changes and improvement to this structure are discussed further in future work.

2.4. Walking Sinusoids Test Model

The "walking sinusoid" model created was an attempt to create a bare-bones model on which to test a hand-coded GA in MATLAB prior to use of a more complex, longer run-time, and more "realistic" CD model. This model was used to investigate GA properties and their effects on convergence, reliability, run-time, and robustness prior to committing to the significantly longer execution time of the "realistic" model.

2.4.1. Walking Sinusoid Model

In theory, The model created is relatively straight forward. A set of parameters meant to represent any combination of sinusoid amplitude, phase, frequency, or noise characteristics to represent the adjustable parameters would be passed to the model from the GA chromosomes after undergoing appropriate conversion and scaling while the "reference" parameters would be hard coded into the model. The model would then generate a sinusoid to represent an

acceleration waveform for both the reference and adjustable parameters and return these waveforms to the GA.

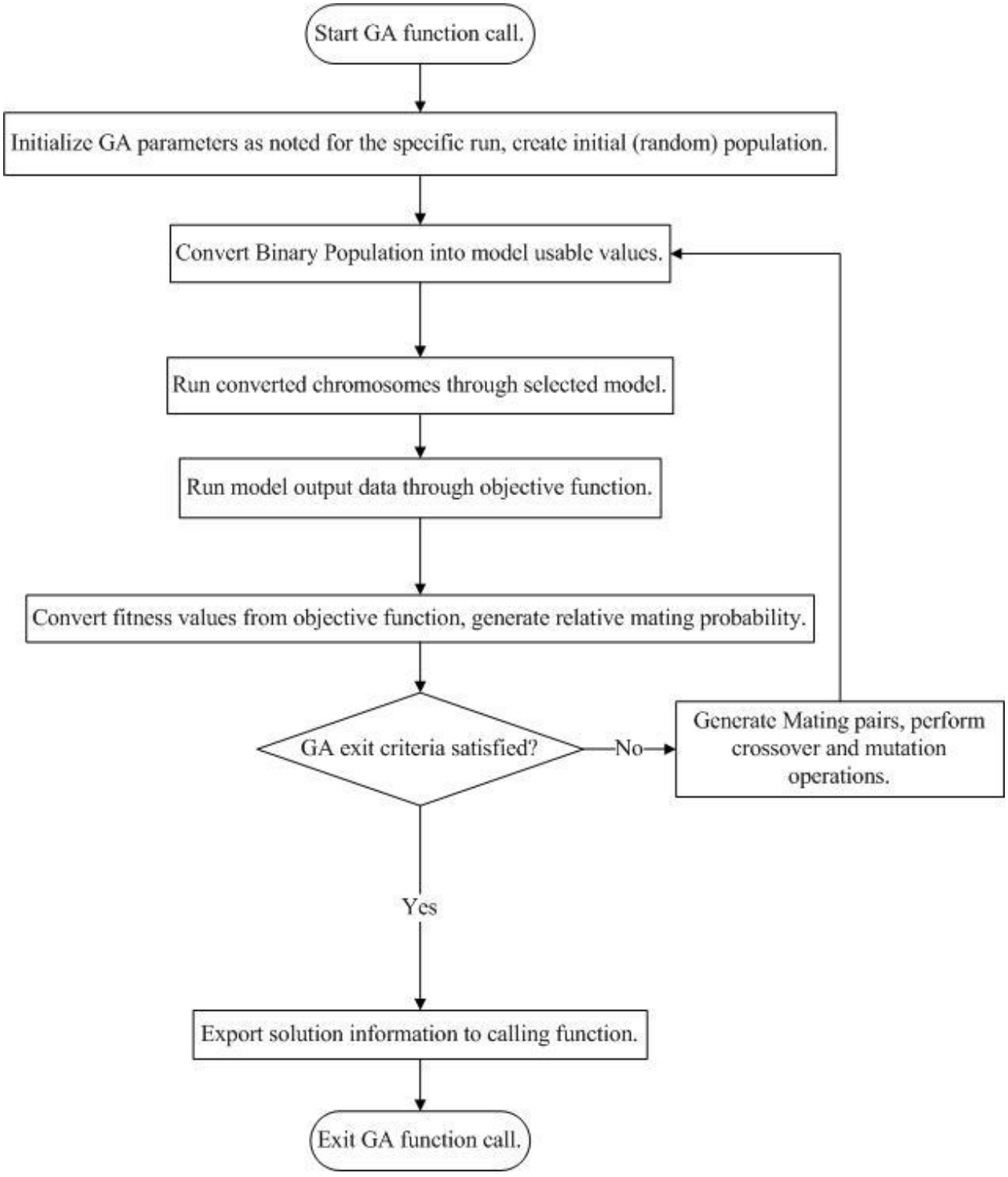

Figure 14: Flow Chart for MATLAB Implemented Genetic Algorithm

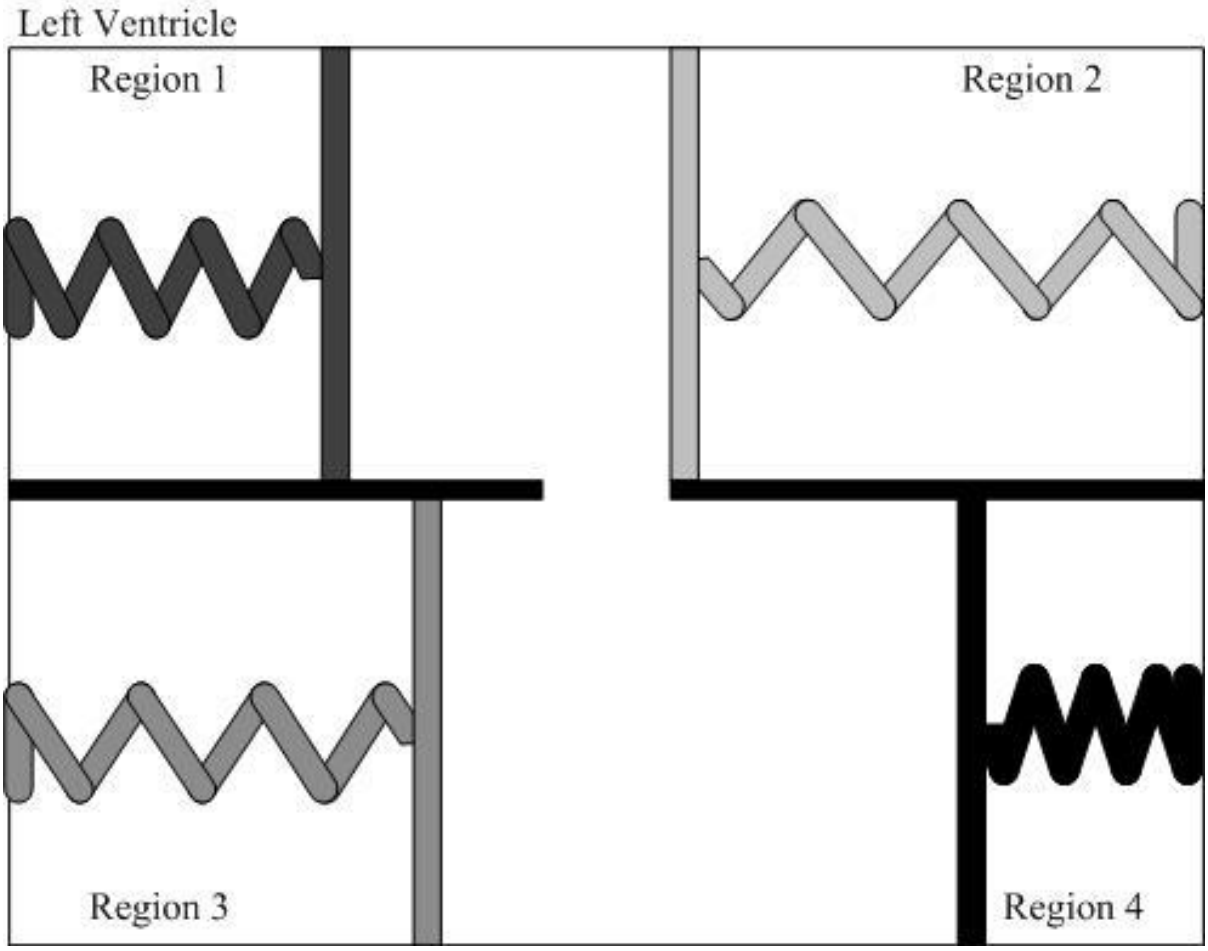

Figure 15: Visual Representation of Cardiac Dyssynchrony Model

2.4.2. GA Operation w/ Walking Sinusoid Model

After starting, the GA would generate a random population of possible solutions as bit strings. These bit strings would be converted and scaled into values usable by the sinusoid model. These converted values would be passed to the model (one chromosome worth) at a time and the returned waveforms would be recorded. Once all chromosomes for the generation are used to generate and record the "accelerations", the recorded waveforms for each chromosome undergo optimization analysis via the sum of differences squared error computations mentioned before. Since a smaller value returned from the optimization function is better, the scores must be "inverted" to compute a proportional fit probability for mating selection. The raw values

returned from the optimization function described in section 2.3 are then normalized by subtracting each raw score from the maximum value for the generation following the equation outlined in section 2.1.5.1.

In doing so, the best fit chromosome receives the largest value, and the worst fit chromosome becomes zero (eliminating its possibility from influencing the next generation). These probabilities are passed to another function developed by [8] to generate the random parings (with replacement) of mates for crossover. Crossover then occurs between the selected chromosomes in the raw (binary) population, and the resulting offspring are copied to the population pool for the next generation and the new population pool is subjected to possible mutation based on the rate selected. The stop conditions are then checked, and if not satisfied, the generation counter is incremented. The cycle then continues with converting and scaling the raw values of the new (now current) generation's population for another pass through the model function, fitness check, mating probability computation/selection, and crossover/mutation, until the stop conditions are satisfied at which point, the GA terminates and returns the best chromosome's values for performance analysis.

2.4.3. Parameter Definitions and Declarations: Walking Sinusoid Model

Several instances of the walking sinusoid/GA combination were executed with variations in noise values, number of optimized parameters, crossover probabilities, etc. Results are outlined in section 5.1. The goal of testing with this model is to determine workable starting points for the GA search parameters prior to implementation of the more complex CD model.

3. EXHAUSTIVE SEARCH

An exhaustive search is a type of objective search function that operates by evaluating every possible solution (in a given, discrete solution space/mapping) with an objective function and comparing the evaluated fitness values to obtain the optimal solution for the given set of choices. Since it tests all possible combinations presented, an exhaustive search is the most reliable way to get as close as possible to the global optimum (minimum in this case) for a search space with unknown contour characteristics. The cost for this reliability is potentially taking a long time to search through solutions. In addition to providing a more definitive answer as to whether variations of timing can improve ventricular acceleration CD, exhaustive searching also gives a "golden standard" to which the GA solutions are compared and conclusions are drawn at the GA's effectiveness in: 1) seeing how well (and consistently) the GA finds a solution close to the solution returned by exhaustive search, and 2) the speed at which the GA returns its solution as compared to an exhaustive search of the entire search space.

3.1. Exhaustive Search Algorithms

To perform the goals outlined previously in section 3, two exhaustive search algorithms were developed in MATLAB to use the same CD model and objective function as the GA uses. These two exhaustive search algorithms are discussed in sections 3.1.1 and 3.1.2.

3.1.1. 2 Parameter Exhaustive Search

Implementing an exhaustive search is a relatively straight forward operation in MATLAB. For the 2 parameter Exhaustive Search, a script file is created to allow the user the ability to select a number of test points between the bounds of the search space. This value is then used to create a vector of test values uniformly spaced between the supplied bounds of the search space. A loop then runs each value in the test vector through the same CD model used by

the GA previously to generate a set of ventricular wall acceleration profiles. Fitness values are then computed for each test point using the same equations as the GA search as shown in section 2.4. The minimum fitness value from these computations identifies the best solution from the exhaustive search. This solution is then also used as the global minimum by which GA determined timing values are compared and evaluated. The final implementation can be found in Appendix E.

3.1.2. 4 Parameter Exhaustive Search

Each time a degree of freedom is added, the number of searches required to test all possible combinations within the specified search space increases exponentially as demonstrated by the following equation:

$$Runs_{Total} = test\ points^{\#\ Parameters}$$

In order to search the space for 3 controlled parameter as required in the 4 parameter search with the same level of resolution in each parameter as in the 2 parameter search, the number of total trials required would be cubed.

Thus, for implementation of the 4 parameter search, slight modifications are made to the search utilized in the 2 parameter case by implementing a hybrid narrowing exhaustive search as shown in Figure 16. To implement the narrowing search for the 4 parameter case, an assumption is made based on experimentation discussed in section 3.2 that the search space is well behaved enough that the initial granularity of the search is sufficient to direct subsequent narrowing to the proper region of convergence in the search space. Some search space surfaces are included in section 5.2.5 showing well behaved surfaces in the case of 2 parameter optimization and more complex surfaces for 4 parameter optimization by investigating the interaction of variation in 2 parameters.

Thus, the 4 parameter exhaustive search allows for user selection of two parameters: the first selects the number of test points within the current bounds of testing, and the second selects the number of times the search zooms, or narrows, those bounds. The resulting search allows for a very fine resolution search with a drastically reduced computation time. While the final search space is larger, the operation of the 4 parameter search follows the basic structure of the 2 parameter search. The search steps through the grid of test points, generates the model acceleration waveforms for those points, computes the cost value by the equation outlined in section 4.2 for the 4 parameter GA search, and maintains the solution for the minimum value as the solution for that trial. The final implementation can be found in Appendix E.

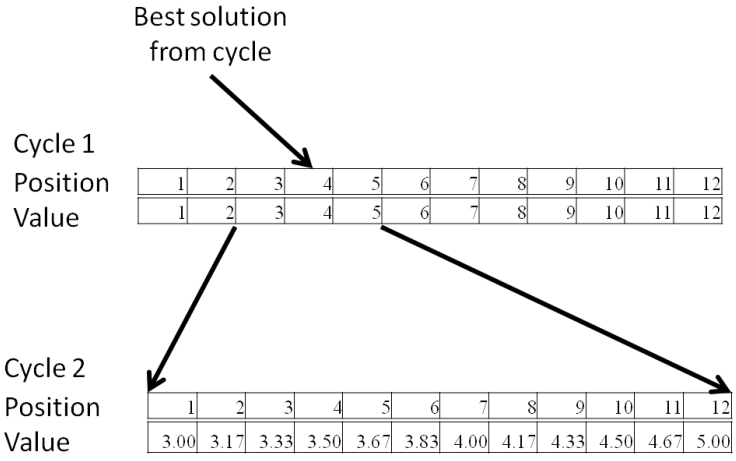

Figure 16: Visualization of 4 Parameter Exhaustive Search Zoom Feature

3.2. Parameter Definitions and Declarations

Parameter values not discussed here, but utilized in exhaustive search optimization algorithms can be found with the included code in Appendix E.

Since the heart rate in the CD model was fixed at 60 beats per minute or 1 beat per second, 1001 test points were chosen for the 2 parameter exhaustive searches to give a search

space of approximately 1 ms resolution with a test point at time = 0 as a starting point for comparison.

As discussed in section 3.1.2, using the same 1001 test points for each parameter as was used in the 2 parameter exhaustive search would create a search space of 1001^3, or 1 billion points. The search time for this is prohibitively large. The solution to this is to use the discussed narrowing search whereby starting from a coarse search grid of 12 points, the algorithm zooms in a total of 6 times and investigates a region of interest further by creating the same number of test points in a region +/- one test point from the best solution for the previous search space. This process is repeated a number of times until the search space becomes small enough to give resolution equal to or exceeding that of using an initial 1001 point search lattice.

From experimentation, the starting grid size to provide reliable convergence is 6 points while 4 points appears to suffer from reliability issues. Since parallel computing was implemented on a 6 core machine to reduce computation time, a multiple of 6 is chosen to maximize the resolution for the same computational efficiency. Further, to provide a buffer zone for better ensuring reliability over the likely complex search spaces investigated in section 5.2.5, 12 points are used with 6 narrowing operations to compute the optimal timings for each of the 4 parameter GA runs.

4. CARDIAC DYSSYNCHRONY MODEL

Note: This model is a first pass approximation of a dyssynchronous left ventricle. This thesis does not claim that the model is adequate or biologically representative of an actual heart, but is rather used as a simplified approximation of cardiac dyssynchrony with the sole purpose of testing the GA optimization of CD across a number of physiologically relevant parameters that can plague a failing heart in a semi-realistic fashion.

To further the cause of determining if timing adjustments hold the power to reduce acceleration CD, a more realistic model of a dyssynchronous heart must be utilized. Like the walking sinusoid model, this model is implemented using MATLAB for use with an exhaustive search algorithm along with the same GA developed and used in testing of the sinusoid model. The same model and optimization function are used with both search algorithms in order to allow parallel investigation of the merits and demerits of each approach.

4.1. "Realistic" Cardiac Dyssynchrony Model

Initial CD model testing is performed by observing 2 of the 4 possible regions in the supplied model. The searches are then scaled to use the full 4 regions possible in order to observe algorithm scaling with more degrees of freedom.

The CD model used was originally developed by a team lead by Dr. Dan Ewert, professor of Electrical and Computer Engineering at NDSU specializing in cardiovascular engineering and research. The team consisted of Sam Oguyemi, McNair scholar and undergraduate student of mechanical engineering at NDSU, and a group of senior design students at Iron Range Engineering for the purpose of creating a graphical user representation of CD for use as a visual aid in classroom instruction on CD and creating an objective set of metrics to measure CD. Assisted by Sam Oguyemi, the model was modified to accept 4 timings relative to a heart beat

from the search algorithm after which, the model calculates and returns the acceleration waveforms of the 4 left ventricular regions over the course of one heart beat. These acceleration waveforms are then used to determine the impact of the specific set of firing times. In short, the model uses a series of differential equations developed, modified, or implemented by the team above in conjunction with built-in MATLAB simultaneous differential equation solvers to generate the required regional wall acceleration values along with the other values necessary for model operation and algorithm evaluation. (See Appendix G for CD Model code and equations utilized.) Values for ventricular tissue parameters such as mechanical resistance, mass, minimum and maximum elastance values, along with heart rate, sample rate, and number of beats prior to returning acceleration measurements can be changed to affect the measured wall accelerations in an attempt to investigate which causes of CD can be corrected by changes in regional firing time.

4.2. GA Operation with "Realistic" CD Model

Operation of the GA occurs much in the same way as it had occurred with the sinusoid model. Runs documented in Tables 3, 4, 9, and 10 along with the corresponding Figures in Appendix A follow the same flow path as that for the Walking Sinusoid model. Differences between the two involve changing the objective function of the 4 parameter GA code to also measure fitness from the two additional control parameters. This process just involves extending the sum squared error function from section 2.4 that operates on one parameter to one that operates on all 3 control values, denoted as N, by the following modification:

$$Cost = \sum_{n=0}^{N} \sum_{k=0}^{K} (ideal(k) - \ experimental_n(k))^2$$

Interpretation of the cost value remains the same (i.e. lower = better). The GA is coded to exit and return the best solution in the population in the event that: the best solution has not improved for 5 successive generations, the objective function value returned is zero, or 50 generations have elapsed.

5. RESULTS

Chapter 5 presents the results of the trials performed broken down by sections based on optimization utilized and parameter deviations. The analysis and interpretation of these results are left to Chapter 6.

5.1. Walking Sinusoids

The walking sinusoid model allows for verification of the GA implementation prior to execution of more computationally intensive models as well as gives a feel for GA noise tolerance. Multiple GA test cases using the Walking Sinusoid model are summarized in Table 1. Figure references (Fig Ref) point to plots that can help visually represent how well the GA was able to match reference parameters in both no noise (NN), and added noise (MN) cases.

Table 1: Walking Sinusoid Model Test Data

		Noise (SD)	# Runs	Min Gen	Max Gen	Run Time	Fig Ref
1 Param	**NN**	0	10	7	14	0.1999 s	F-1, F-2
	MN	2	10	29	713	7.9309 s	F-3, F-4
2 Param	**NN**	0	10	20	49	1.1563 s	F-5, F-6
	MN	2	10	24	1493	31.1175 s	F-7, F-8

5.2. "Realistic" CD Model

Several instances of the "Realistic" CD model are investigated with variations in the values associated with the possible mechanical variations between ventricular regions implemented in the CD model. Each test case is executed and evaluated using GA optimization, ES optimization, and no optimization/adjustment (where all timings are fixed and equal). Results from each test case are tabulated here in tables corresponding to one of the three optimization theories. Additionally, acceleration and timing data for all test cases are plotted in Appendices A, B, and C for a visual representation as noted by the respective figure reference

column for each test case. To gain an idea of the baseline performance expected of each search type prior to operating the model in a state where the expected results are unknown, three configurations are attempted for the no dyssynchrony GA setup: the first configuration sets 3 of the 4 regions to a known timing value and attempts to find the 4^{th} time, the second configuration sets 2 of the 4 regions to a known timing value and ties the other 2 regions together (effectively creating 2 larger regions) and attempts to find a time for the 2^{nd}, large region. Finally, the third configuration sets one of the 4 regions and attempts to find individual timings for the other 3 regions. Initial results show the first 2 test configurations outlined here as operating almost identically. Thus, in subsequent runs for the various dyssynchronous conditions, only test configurations 1 and 3 from the no dyssynchrony set are replicated for each test case.

5.2.1. Select Trial Data

This section shows an extended data set for a small sample of the numerous trials performed. These trials use modified CD model initial conditions to alleviate computational problems found with the model as discussed in section 6.2.

As is discussed in section 5.2.5, GA trials do not yet perform to levels that can compete with exhaustive searches in either computational time or level of fitness when utilizing more than 2 parameters. Thus, these select trials focus on, and draw conclusions from, exhaustive search optimization data sets. Table 2 gives data for two sets of trial conditions using both old and new model parameters. Figures 17 and 18 show pressure-volume, PV, loops for each set of trial conditions using the new model with the values generated from no timing variation overlaid on the set optimized by exhaustive search. Figures 19 and 20 show Wiggers diagrams outlining pressure, volume, and regional timing comparisons for the same conditions as in Figures 17 and 18. Figures 21 and 22 show acceleration waveforms and corresponding regional timings for the

no dyssynchrony (ND) trial in Table 2 for both exhaustive search (ES) and no timing adjustment (NA) trials respectively. Figures 23 and 24 show the same acceleration and timing information as Figures 20 and 21 but for the maximum elastance ($E_{max}D$) trial. Figures 25 – 28 show zoomed, normalized regional timing data for each of the trials depicted in Figures 21 – 24 respectively. Figures 29 – 32 show regional work performed, and Figures 33 – 36 show regional instantaneous power for the respective conditions of Figures 21 – 24.

Table 2: Trial Data, New vs. Old CD Model Parameters

ES Trial		AP_2	t_{ES2}	t_{ES3}	t_{ES4}	EF	%Δ EF	CO_{ES}	%Δ CO	%Δ F	Fig Ref
ES, ND, N	**(1,3)**	N/A	1.0000	1.0000	1.0000	0.4764	0.0000	3469	0.0000	2.98E+08	16, 18, 20, 24, 28, 32
ES, ND, O	**(1,3)**	N/A	1.0000	1.0000	1.0000	0.3782	0.0000	1450	0.0000	3.42E+09	B-5, B-6
NA, ND, N	**(1,3)**	N/A	1.0000	1.0000	1.0000	0.4764	N/A	3469	N/A	N/A	16, 18, 21, 25, 29, 33
NA, ND, O	**(1,3)**	N/A	1.0000	1.0000	1.0000	0.3782	N/A	1450	N/A	N/A	C-5, C-6
ES, $E_{max}D$, N	**(1,3)**	40	0.8100	0.9871	0.9872	0.4982	5.3945	3532	0.0275	-5.62E+01	17, 19, 22, 26, 30, 34
ES, $E_{max}D$, O	**(1,3)**	40	0.8364	1.0085	1.0082	0.3959	0.6611	1495	-0.4251	-3.01E+01	B-21, B-22
NA, $E_{max}D$, N	**(1,3)**	40	1.0000	1.0000	1.0000	0.4727	N/A	3531	N/A	N/A	17, 19, 23, 27, 31, 35
NA, $E_{max}D$, O	**(1,3)**	40	1.0000	1.0000	1.0000	0.3933	N/A	1502	N/A	N/A	C-21, C-22

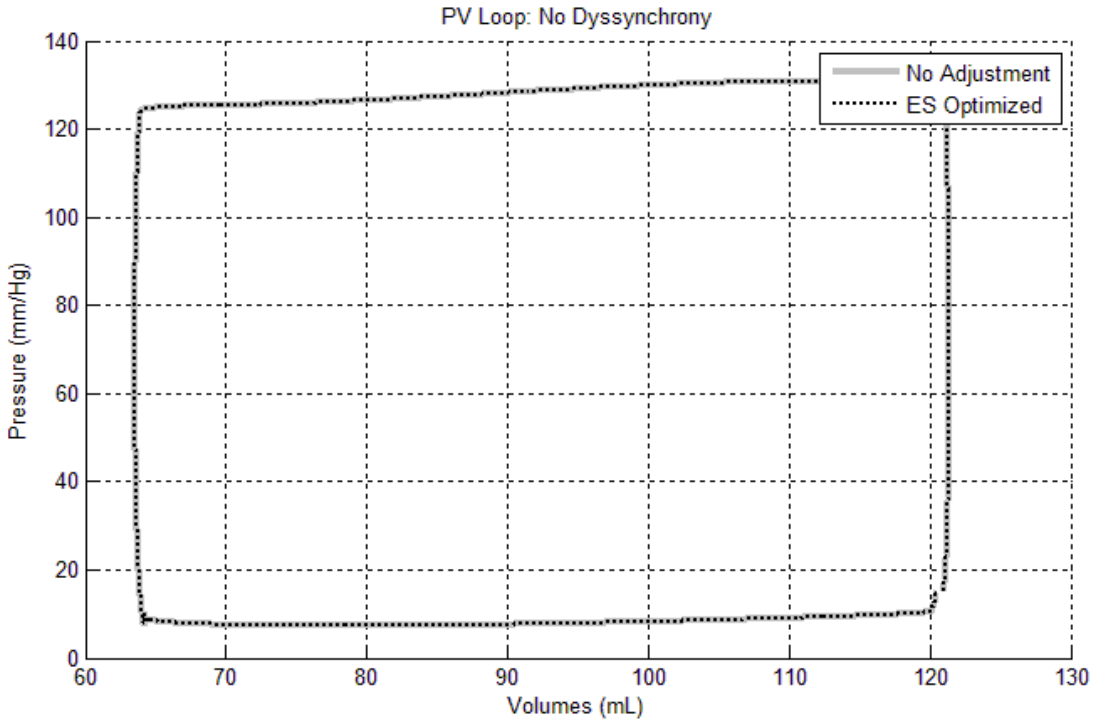

Figure 17: PV Loop, No Dyssynchrony, ES vs. No Adjustment

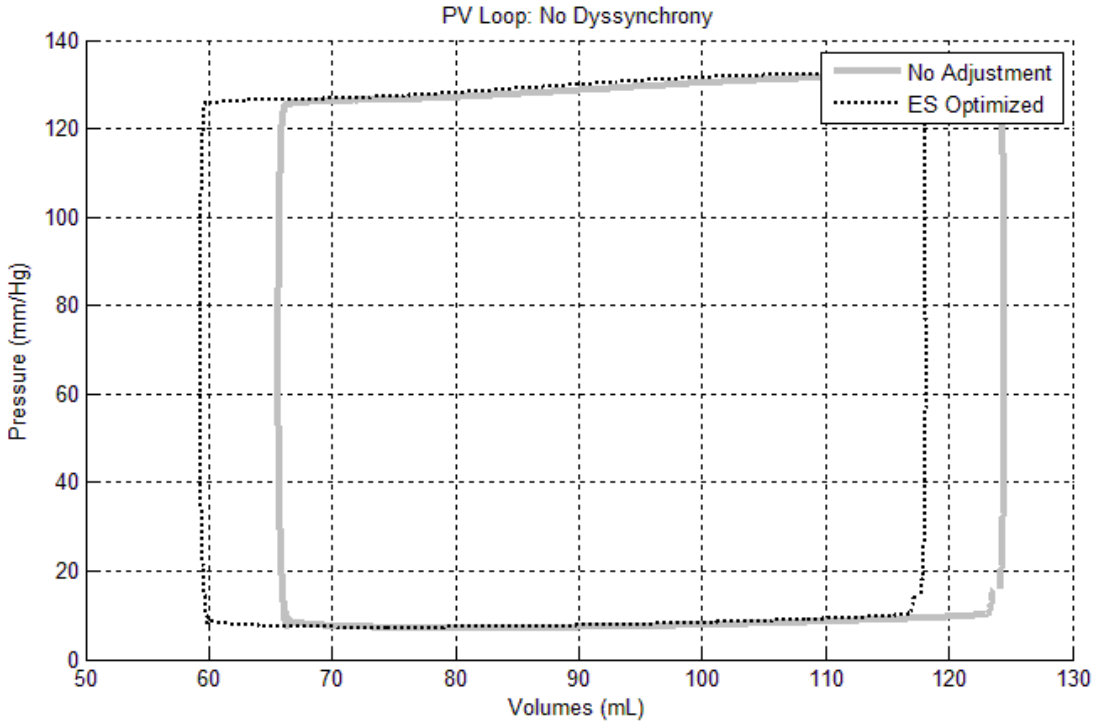

Figure 18: PV Loop, Maximum Elastance Dyssynchrony, ES vs. No Adjustment

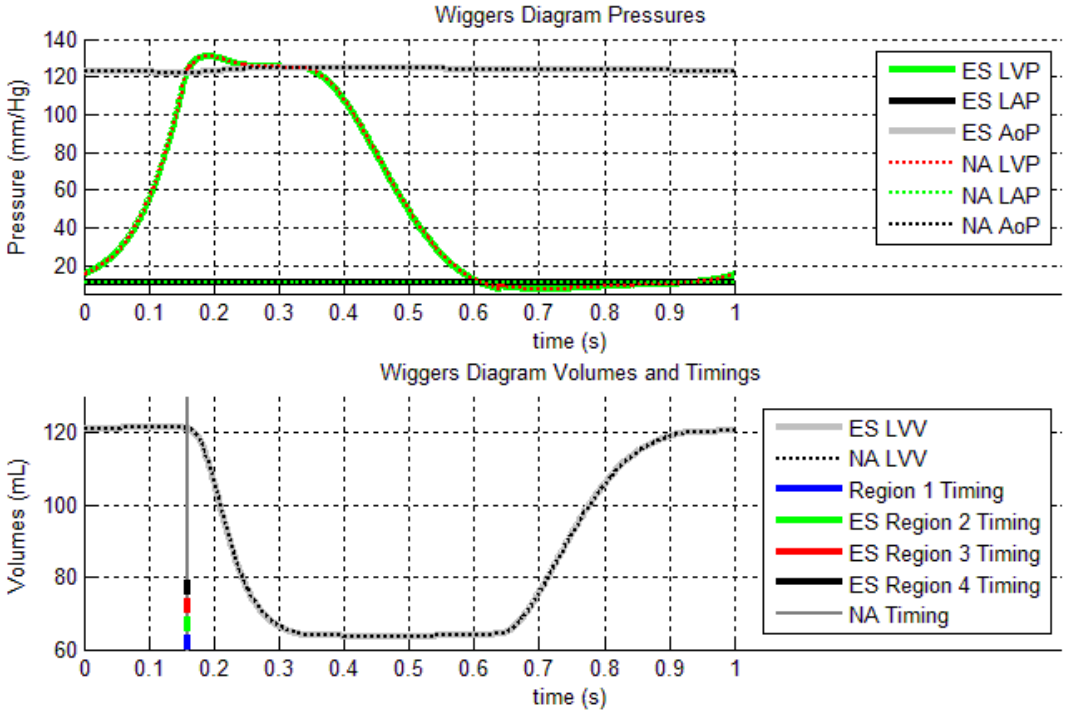

Figure 19: Wiggers Diagram, No Dyssynchrony, ES vs. No Adjustment

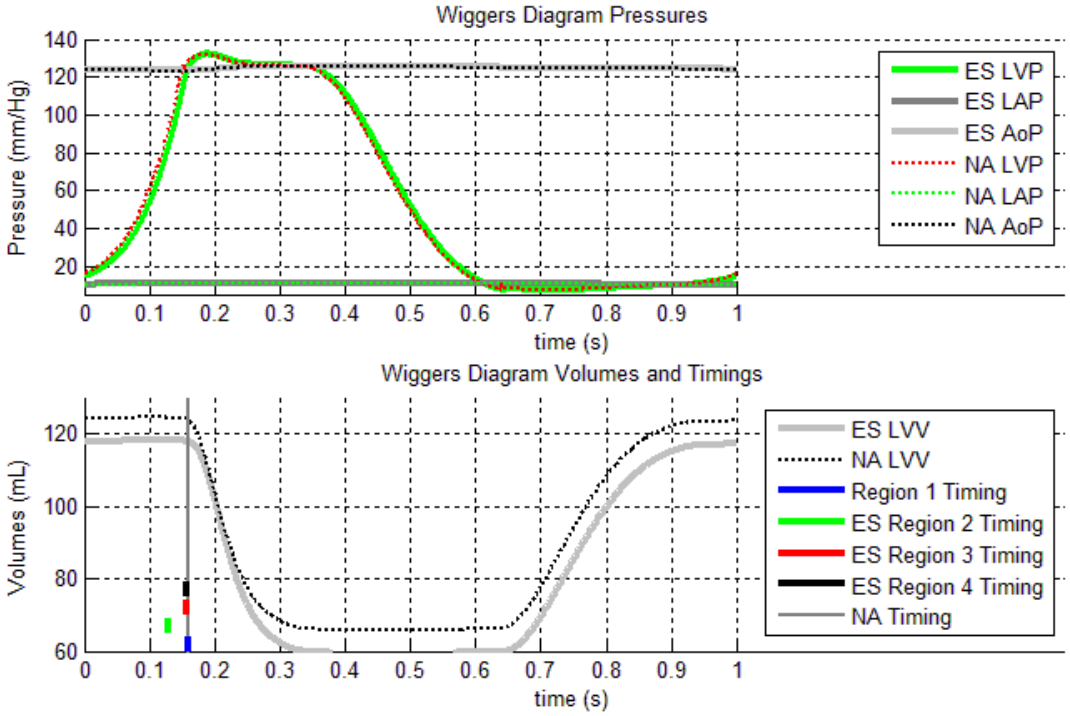

Figure 20: Wiggers Diagram, Maximum Elastance Dyssynchrony, ES vs. No Adjustment

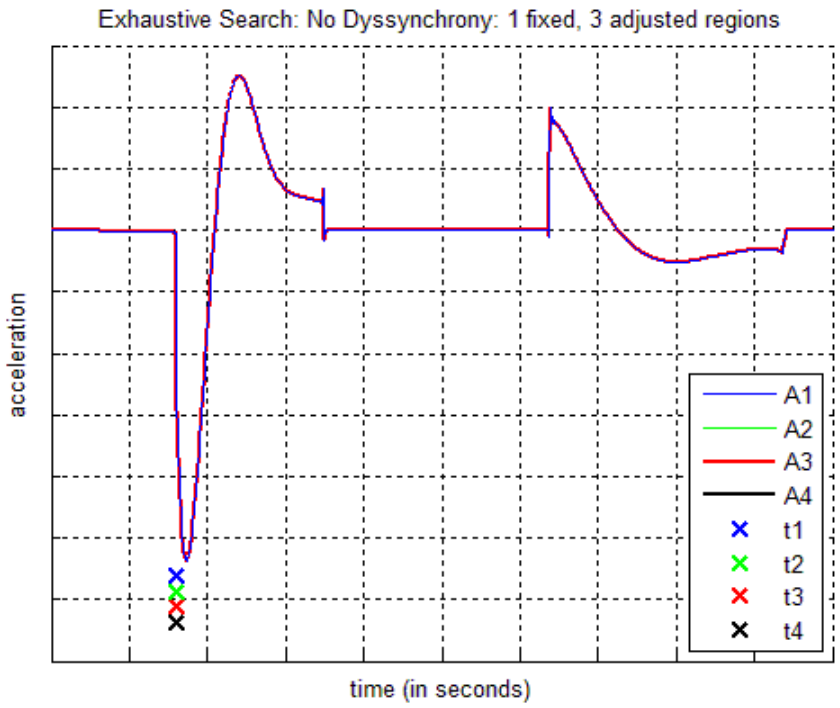

Figure 21: Acceleration Plot with Firing Times, ES, ND (1,3), New CD Model

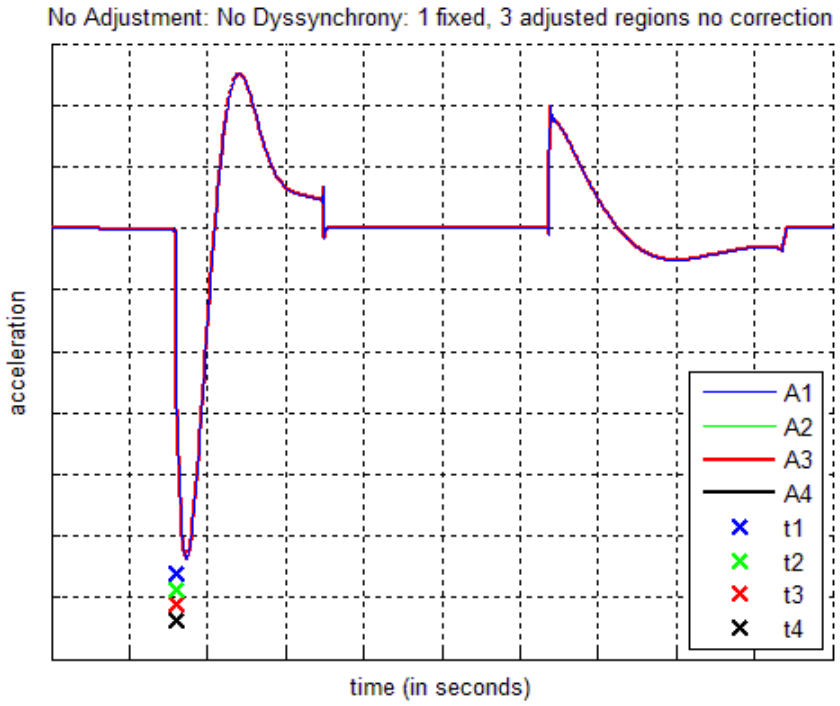

Figure 22: Acceleration Plot with Firing Times, NA, ND (1,3), New CD Model

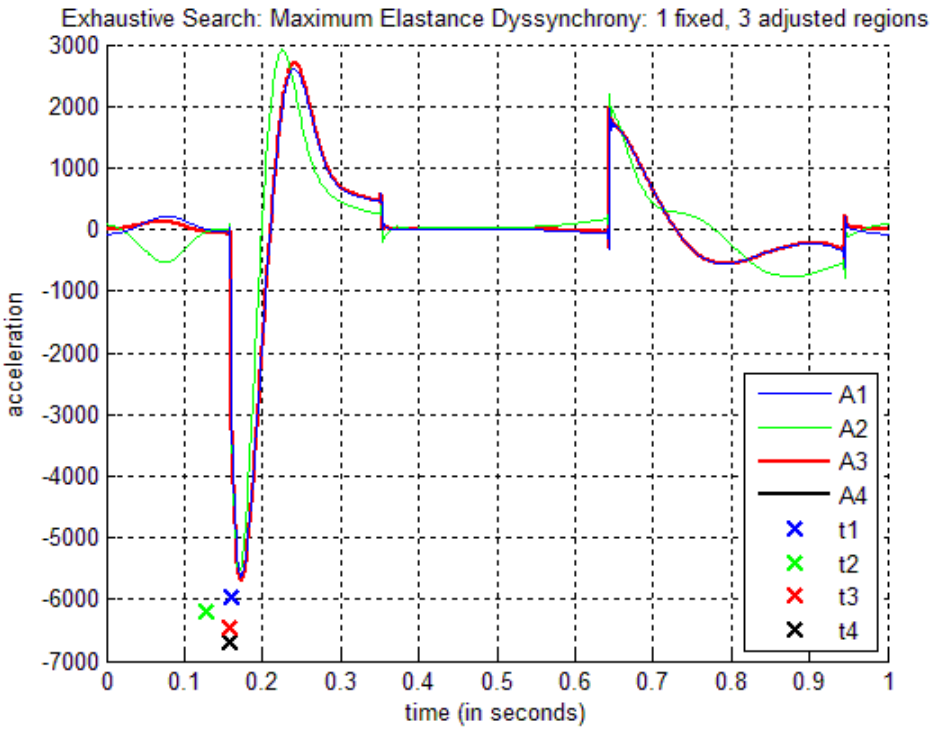

Figure 23: Acceleration Plot with Firing Times, ES, $E_{max}D$ (1,3), New CD Model

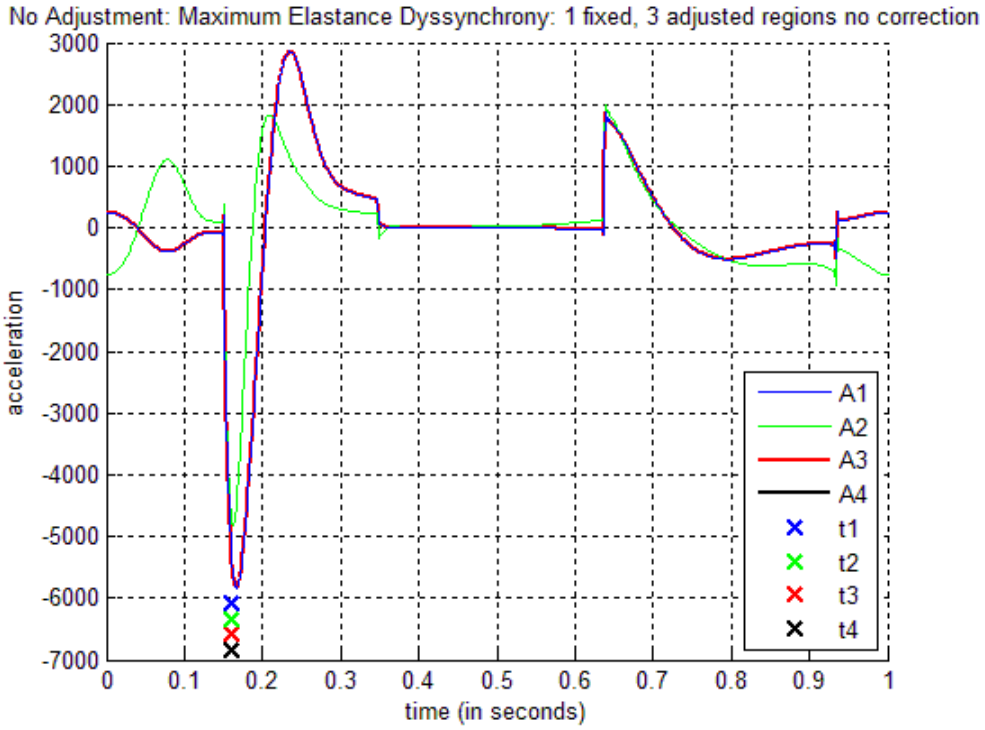

Figure 24: Acceleration Plot with Firing Times, NA, EmaxD (1,3), New CD Model

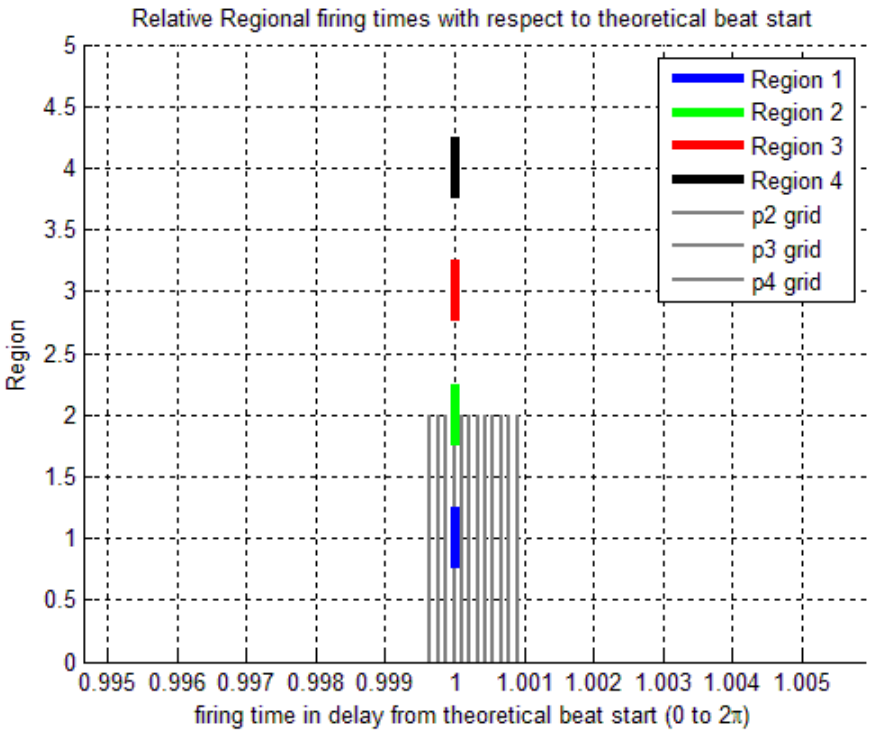

Figure 25: Zoomed Firing Times, ES, ND (1,3), New CD Model

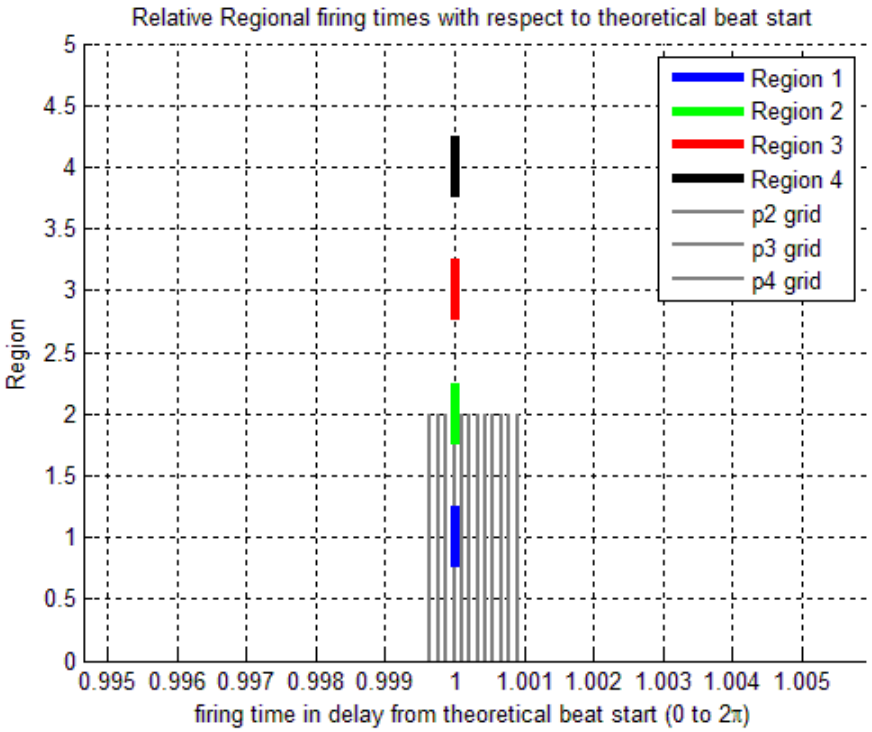

Figure 26: Zoomed Firing Times, NA, ND (1,3), New CD Model

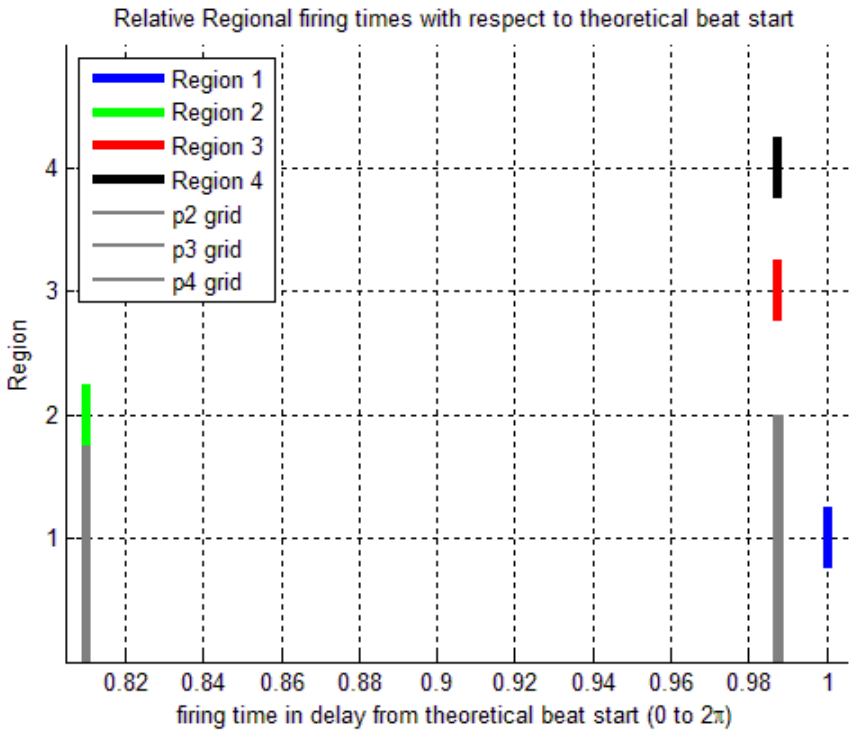

Figure 27: Zoomed Firing Times, ES, EmaxD (1,3), New CD Model

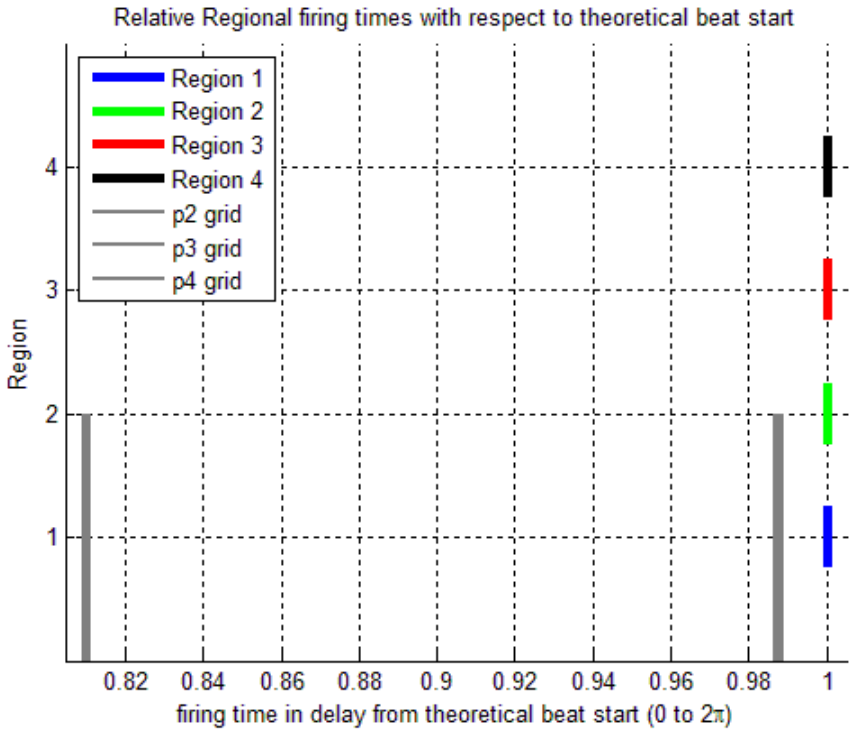

Figure 28: Zoomed Firing Times, NA, EmaxD (1,3), New CD Model

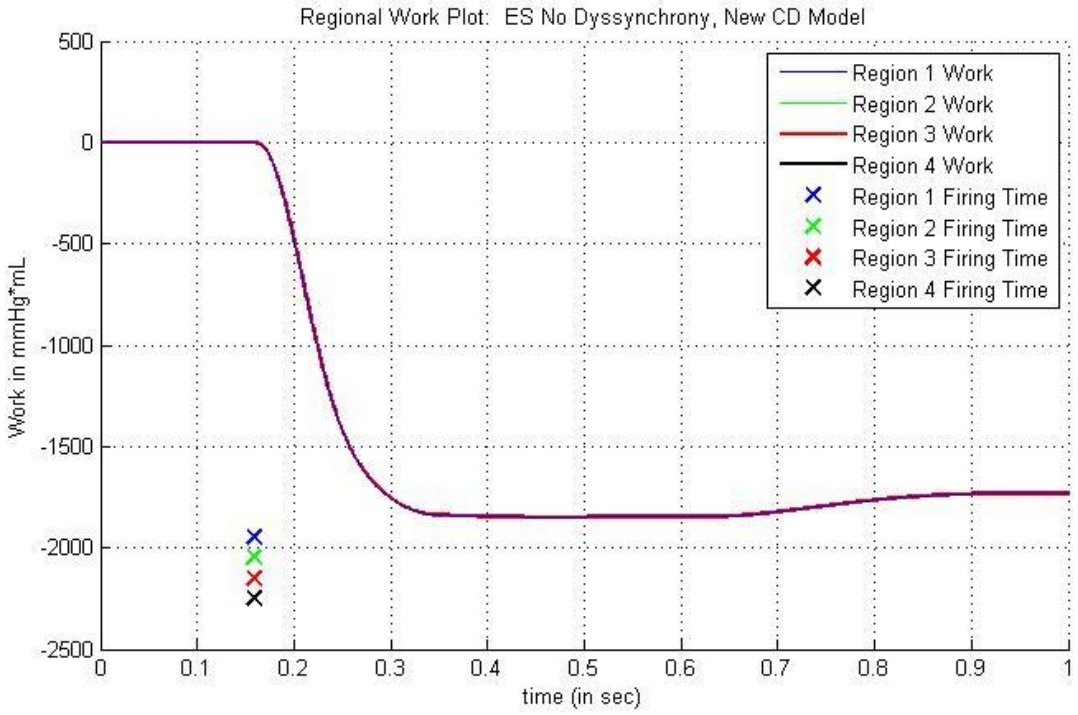

Figure 29: Work Plot with Firing Times, ES, ND (1,3), New CD Model

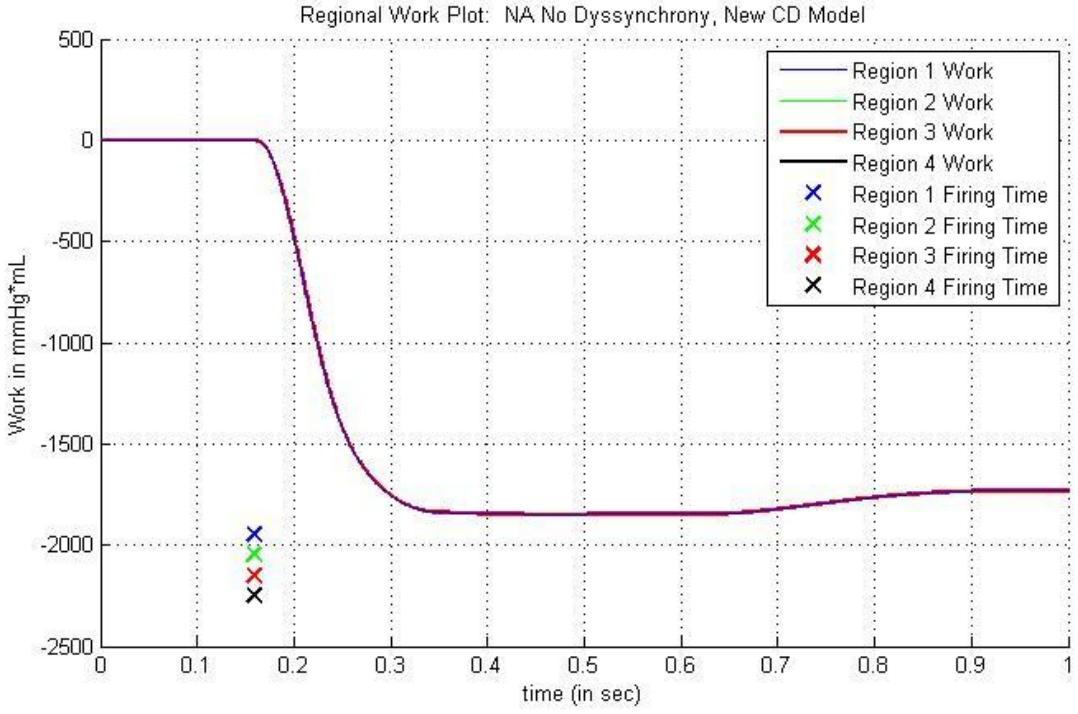

Figure 30: Work Plot with Firing Times, NA, ND (1,3), New CD Model

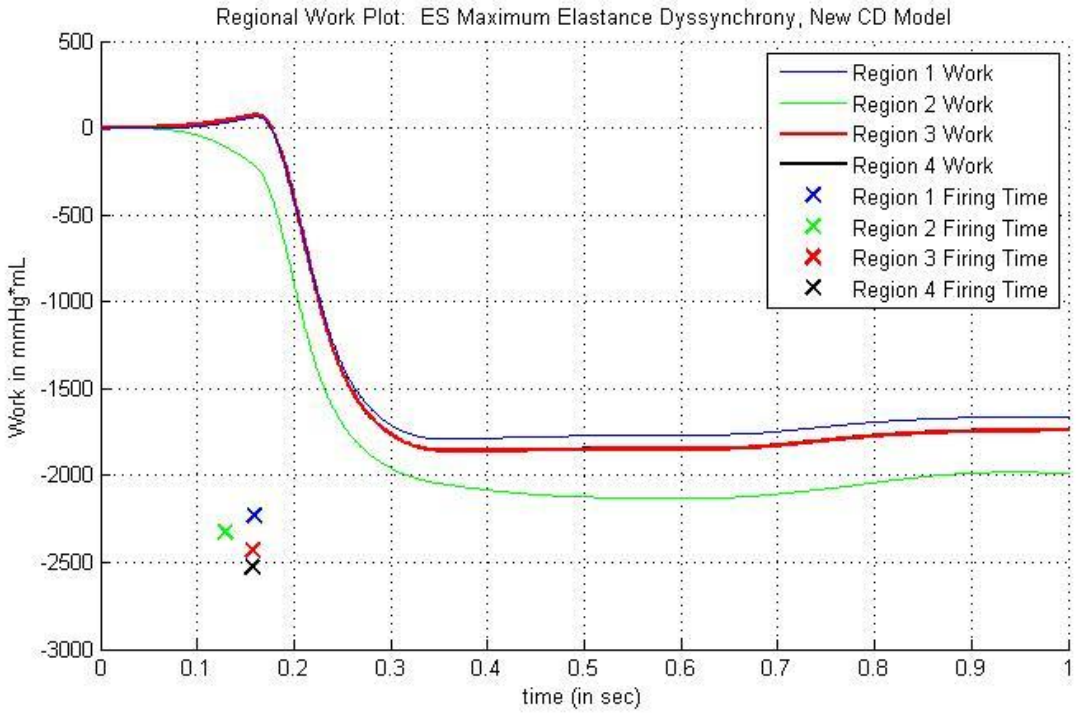

Figure 31: Work Plot with Firing Times, ES, EmaxD (1,3), New CD Model

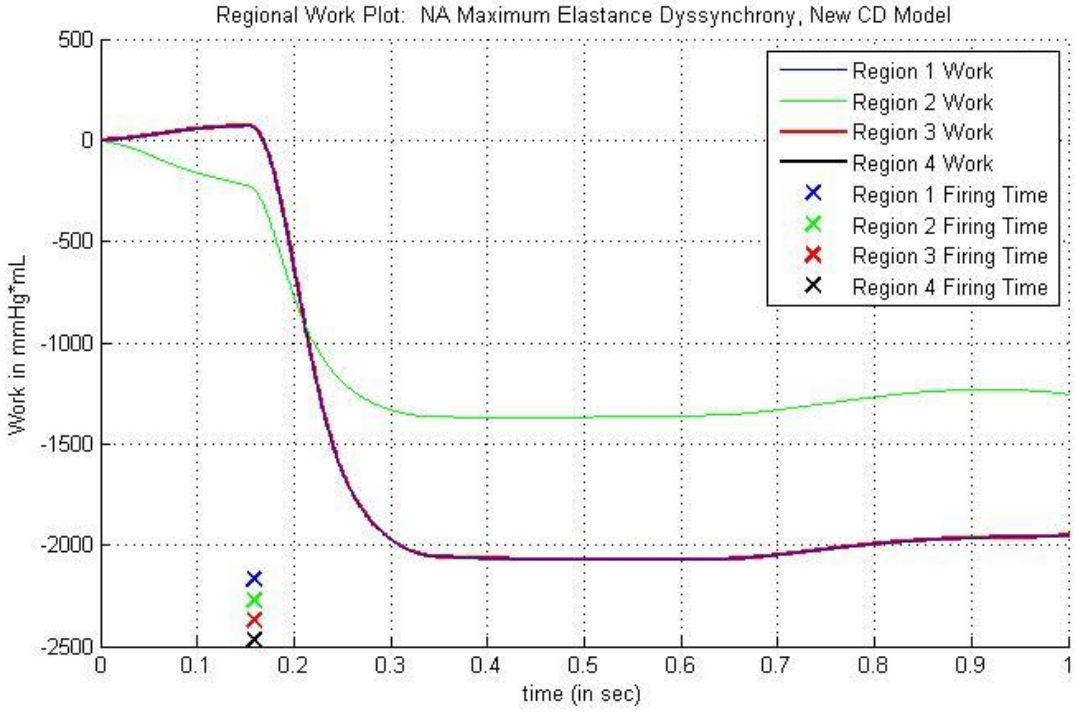

Figure 32: Work Plot with Firing Times, NA, EmaxD (1,3), New CD Model

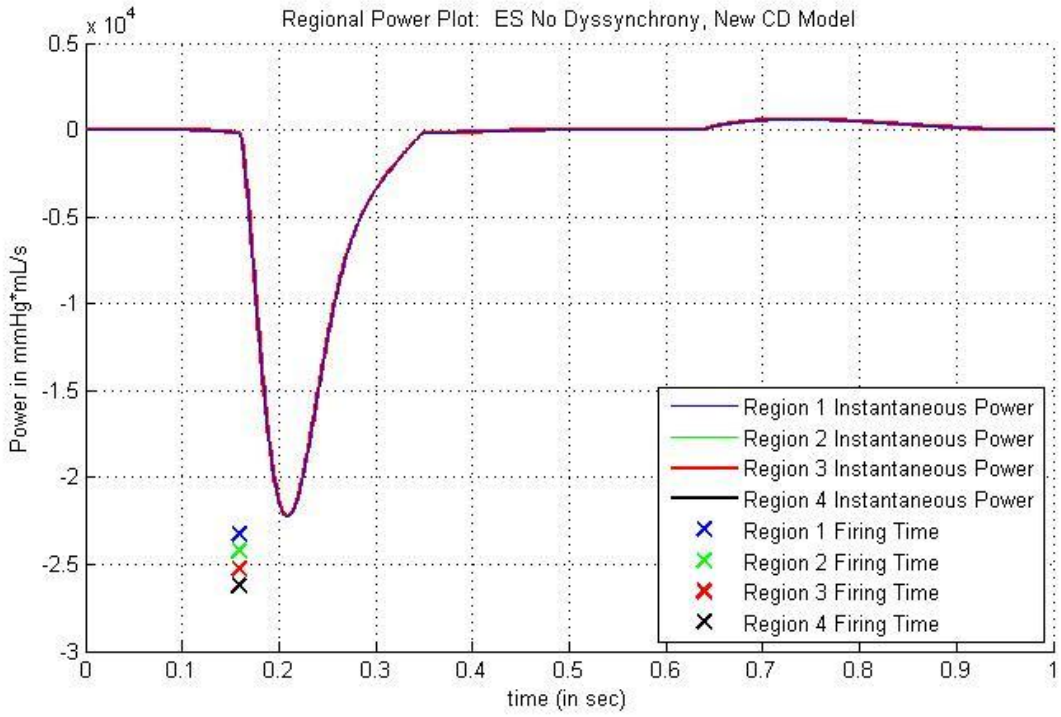

Figure 33: Instantaneous Power Plot with Firing Times, ES, ND (1,3), New CD Model

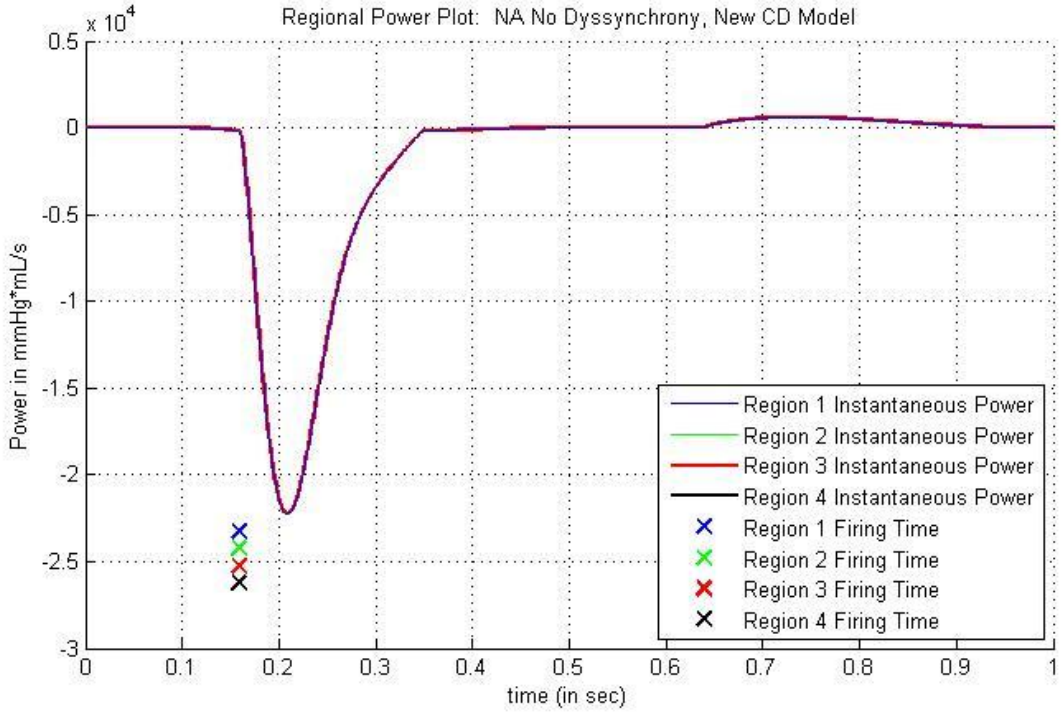

Figure 34: Instantaneous Power Plot with Firing Times, NA, ND (1,3), New CD Model

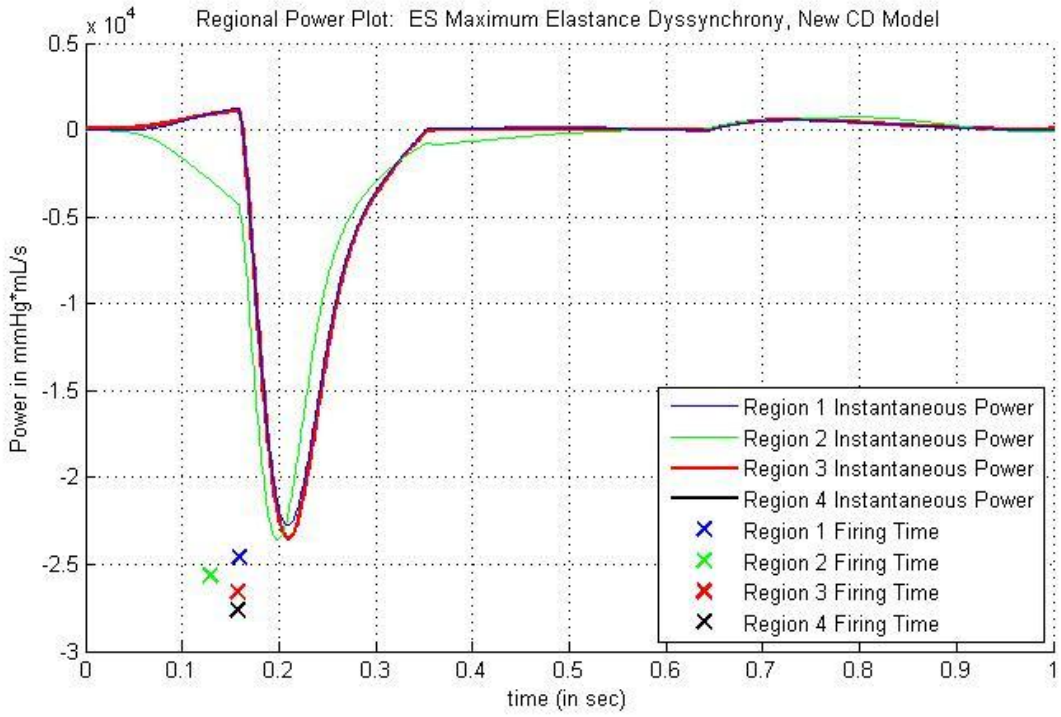

Figure 35: Instantaneous Power Plot with Firing Times, ES, EmaxD (1,3), New CD Model

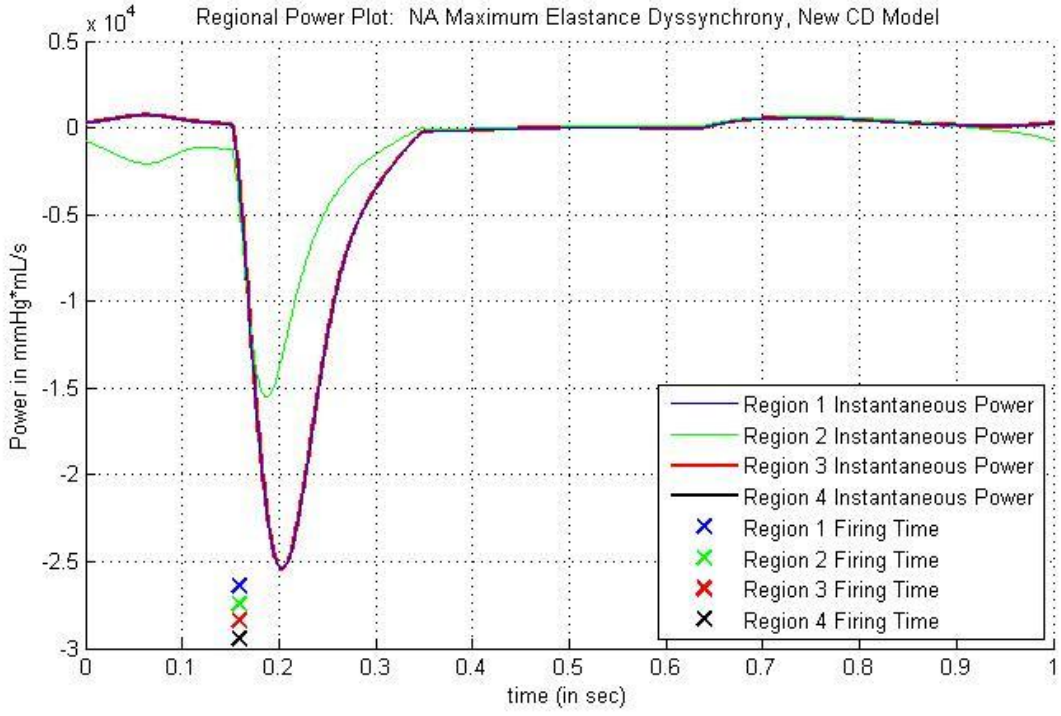

Figure 36: Instantaneous Power Plot with Firing Times, NA, EmaxD (1,3), New CD Model

5.2.2. GA Search Cases: Tables

Table 3: 2 Parameter GA Search Case Results for CD Model

GA Trial		# Runs	AP_2	t_{Nmin2}	t_{Navg2}	t_{Nmax2}	t_{NES}	EF_{GA}	EF_{ES}	%Δ EF
ND	**(3,1)**	10	N/A	0.4955	0.9827	1.3284	0.99900	0.37850	0.37820	0.0793
	(2,2)	10	N/A	0.7578	0.9647	1.3300	0.99900	0.37940	0.37820	0.3173
RD	**(3,1)**	10	0.0150	0.5614	0.9112	1.0385	0.99900	0.35070	0.35150	-0.2276
MD	**(3,1)**	10	0.0100	0.7578	1.0212	1.3422	1.04300	0.39590	0.39460	0.3294
$E_{min}D$	**(3,1)**	10	4.0000	0.7578	1.0298	1.3468	1.00530	0.32440	0.32440	0.0000
$E_{max}D$	**(3,1)**	10	40.0000	0.5614	0.8089	0.8958	0.83570	0.39660	0.39630	0.0757

GA Trial cont.		CO_{GA}	CO_{ES}	%Δ CO	%Δ F	Fig Ref
ND	**(3,1)**	1450	1450	-0.0012	1.61E+04	A-1, A-2
	(2,2)	1450	1450	-0.0058	6.66E+04	A-3, A-4
RD	**(3,1)**	1312	1318	-0.4481	-5.25E-01	A-7, A-8
MD	**(3,1)**	1528	1532	-0.3043	2.68E+00	A-11, A-12
$E_{min}D$	**(3,1)**	1207	1207	0.0075	6.06E-01	A-15, A-16
$E_{max}D$	**(3,1)**	1494	1496	-0.1099	1.24E+00	A-19, A-20

Table 4: 4 Parameter GA Search Case Results for CD Model

GA Trial		# Runs	AP_2	t_{Nmin2}	t_{Navg2}	t_{Nmax2}	t_{Nmin3}	t_{Navg3}	t_{Nmax3}	t_{Nmin4}	t_{Navg4}	t_{Nmax4}	t_{NES2}	t_{NES3}	t_{NES4}
ND	**(1,3)**	10	N/A	0.8345	1.2907	2.0417	0.1580	0.8897	1.4235	0.7348	1.2056	1.9696	1.0000	1.0000	1.0000
RD	**(1,3)**	10	0.0150	0.7593	1.1586	1.4036	0.5262	1.4176	3.1477	0.6059	2.2849	5.6082	0.9461	0.9968	0.9963
MD	**(1,3)**	10	0.0100	0.3712	1.1152	1.4726	0.0276	2.1790	5.6266	0.7670	3.1442	5.7892	1.0412	1.0000	0.9994
$E_{min}D$	**(1,3)**	10	4.0000	0.5446	1.0744	1.7763	0.1580	0.9560	2.0126	0.6044	2.2355	6.0792	1.0111	1.0043	1.0045
$E_{max}D$	**(1,3)**	10	40.0000	0.7854	0.9633	1.3744	0.1580	1.3660	3.4146	0.5630	1.6783	2.8363	0.8364	1.0085	1.0082

GA Trial cont.		EF_{GA}	EF_{ES}	%Δ EF	CO_{GA}	CO_{ES}	%Δ CO	%Δ F	Fig Ref
ND	**(1,3)**	0.3642	0.3782	-3.7017	1432	1450	-1.2788	1.92E+10	A-5, A-6
RD	**(1,3)**	0.2669	0.3501	-23.7646	19	1308	-98.5788	1.34E+03	A-9, A-10
MD	**(1,3)**	0.1156	0.3947	-70.7119	475	1523	-68.8217	1.50E+03	A-13, A-14
$E_{min}D$	**(1,3)**	0.2587	0.3243	-20.2282	1031	1207	-14.5487	3.91E+03	A-17, A-18
$E_{max}D$	**(1,3)**	0.3586	0.3959	-9.4216	1457	1495	-2.5724	8.52E+02	A-21, A-22

5.2.3. Exhaustive Search Cases: Tables

Table 5: 2 Parameter Exhaustive Search Case Results for CD Model

ES Trial		AP_2	t_{ES2}	EF_{ES}	EF_{NA}	%Δ EF	CO_{ES}	CO_{NA}	%Δ CO	%Δ F	Fig Ref
ND	**(3,1)**	N/A	0.9990	0.3782	0.3782	0.0000	1450	1450	0.0012	1.28E+13	B-1, B-2
	(2,2)	N/A	0.9990	0.3782	0.3782	0.0000	1450	1450	0.0004	1.28E+13	B-3, B-4
RD	**(3,1)**	0.0150	0.9990	0.3515	0.3515	0.0000	1318	1318	-0.0137	-5.19E-02	B-7, B-8
MD	**(3,1)**	0.0100	1.0430	0.3946	0.3972	-0.6546	1523	1532	-0.6015	-7.26E+00	B-11, B-12
$E_{min}D$	**(3,1)**	4.0000	1.0053	0.3244	0.3245	-0.0308	1207	1207	-0.0144	-7.08E-02	B-15, B-16
$E_{max}D$	**(3,1)**	40.0000	0.8357	0.3963	0.3933	0.76278	1496	1502	-0.4027	-2.98E+01	B-19, B-20

Table 6: 4 Parameter Exhaustive Search Case Results for CD Model

ES Trial		AP_2	t_{ES2}	t_{ES3}	t_{ES4}	EF_{ES}	EF_{NA}	%Δ EF	CO_{ES}	CO_{NA}	%Δ CO	%Δ F	Fig Ref
ND	**(1,3)**	N/A	1.0000	1.0000	1.0000	0.3782	0.3782	0.0000	1450	1450	0.0000	3.42E+09	B-5, B-6
RD	**(1,3)**	0.0150	0.9461	0.9968	0.9963	0.3501	0.3515	-0.3983	1308	1318	-0.7289	-1.42E+00	B-9, B-10
MD	**(1,3)**	0.0100	1.0412	1.0000	0.9994	0.3947	0.3972	-0.6294	1523	1532	-0.5807	-7.38E+00	B-13, B-14
$E_{min}D$	**(1,3)**	4.0000	1.0111	1.0043	1.0045	0.3243	0.3245	-0.0616	1207	1207	0.0045	-4.01E-01	B-17, B-18
$E_{max}D$	**(1,3)**	40.0000	0.8364	1.0085	1.0082	0.3959	0.3933	0.6611	1495	1502	-0.4251	-3.01E+01	B-21, B-22
R_2D	**(1,3)**	See Fig	0.9757	0.9844	0.985	0.0568	0.0565	0.5310	1302	1302	0.0207	-1.18E+00	B-23, B-24
$R_{234}D$	**(1,3)**	See Fig	1.0236	1.0304	1.0954	0.0293	0.0294	-0.3401	782	783	-0.1954	-3.20E+00	B-25, B-26

5.2.4. No Timing Adjustment: Tables

Table 7: 2 Parameter No Adjustment Case Results for CD Model

NA Trial		AP_2	t_2	EF_{NA}	CO_{NA}	Fig Ref
ND	**(3,1)**	N/A	1.0000	0.3782	1450	C-1, C-2
	(2,2)	N/A	1.0000	0.3782	1450	C-3, C-4
RD	**(3,1)**	0.0150	1.0000	0.3515	1318	C-7, C-8
MD	**(3,1)**	0.0100	1.0000	0.3972	1532	C-11, C-12
$E_{min}D$	**(3,1)**	4.0000	1.0000	0.3245	1207	C-15, C-16
$E_{max}D$	**(3,1)**	40.0000	1.0000	0.3933	1502	C-19, C-20

Table 8: 4 Parameter No Adjustment Case Results for CD Model

NA Trial		AP_2	t_2	t_3	t_4	EF_{NA}	CO_{NA}	Fig Ref
ND	**(1,3)**	N/A	1.0000	1.0000	1.0000	0.3782	1450	C-5, C-6
RD	**(1,3)**	0.0150	1.0000	1.0000	1.0000	0.3515	1318	C-9, C-10
MD	**(1,3)**	0.0100	1.0000	1.0000	1.0000	0.3972	1532	C-13, C-14
$E_{min}D$	**(1,3)**	4.0000	1.0000	1.0000	1.0000	0.3245	1207	C-17, C-18
$E_{max}D$	**(1,3)**	40.0000	1.0000	1.0000	1.0000	0.3933	1502	C-21, C-22
R_2D	**(1,3)**	See Fig	1.0000	1.0000	1.0000	0.0565	1302	C-23, C-24
$R_{234}D$	**(1,3)**	See Fig	1.0000	1.0000	1.0000	0.0294	783	C-25, C-26

5.2.5. Optimization Function Fitness Surface

Optimization function fitness surface geometry is investigated via exhaustive search in order to create a map of fitness values in both 1 and 2 varying parameters. The reasons for this are 3 fold. First, creating maps for two sets of 1 varying parameters can show that each model region is computed and treated equally. Second, the map for 1 varying parameter shows the level of impact of changing the timing of a single region when the other regions are held equal and constant. Third, creating the map for 2 varying parameters can show how having up to 3 different timing values present in the model causes the regions to interact with each other and gives scaled intuition and insight to how the surface likely changes when increasing the number of varying parameters further (3+ varying regions). Due to the dimensionality of the fitness surface, variation in more than 2 regions simultaneously cannot be visualized graphically. Figures 37 and 38 show fitness values for timing variation in region 3 using both standard and logarithmic (base 10) scales respectively while Figures 38 and 39 depict the same results for region 4. Additionally, Figures 41 and 42 show the results of two simultaneously time varying regions (regions 3 and 4) with standard scale in both 3d and top views. Figures 43 and 44 depict the same 3d and top views using a base 10 logarithmic scales for cost.

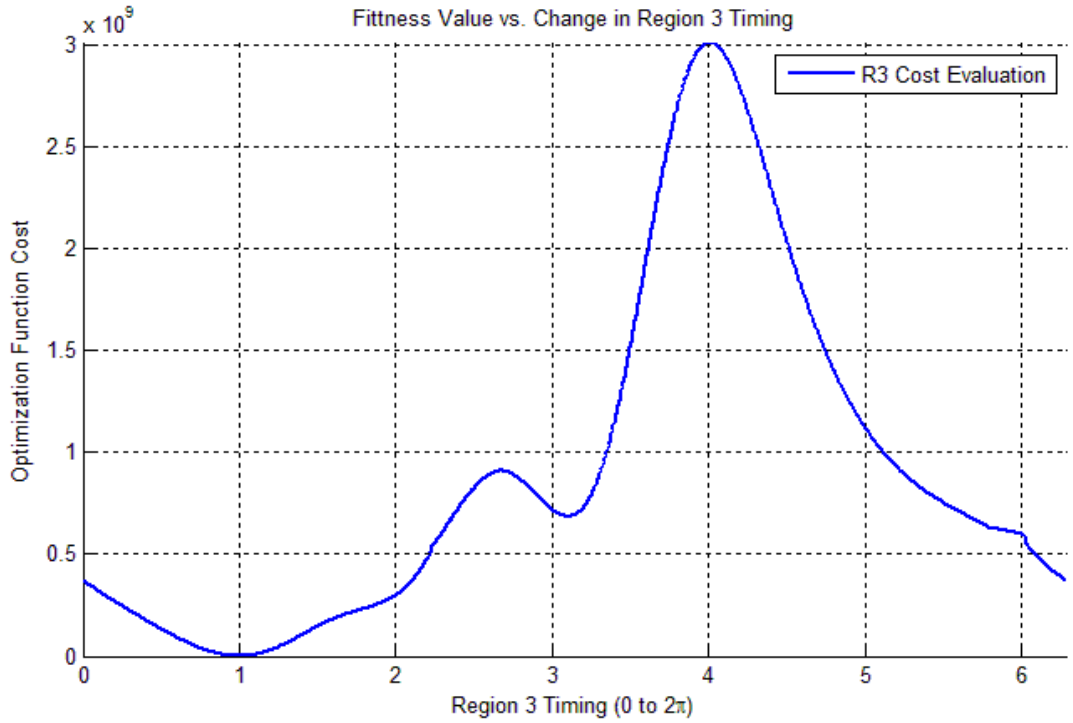

Figure 37: Cost Curve for Region 3 Timing Changes

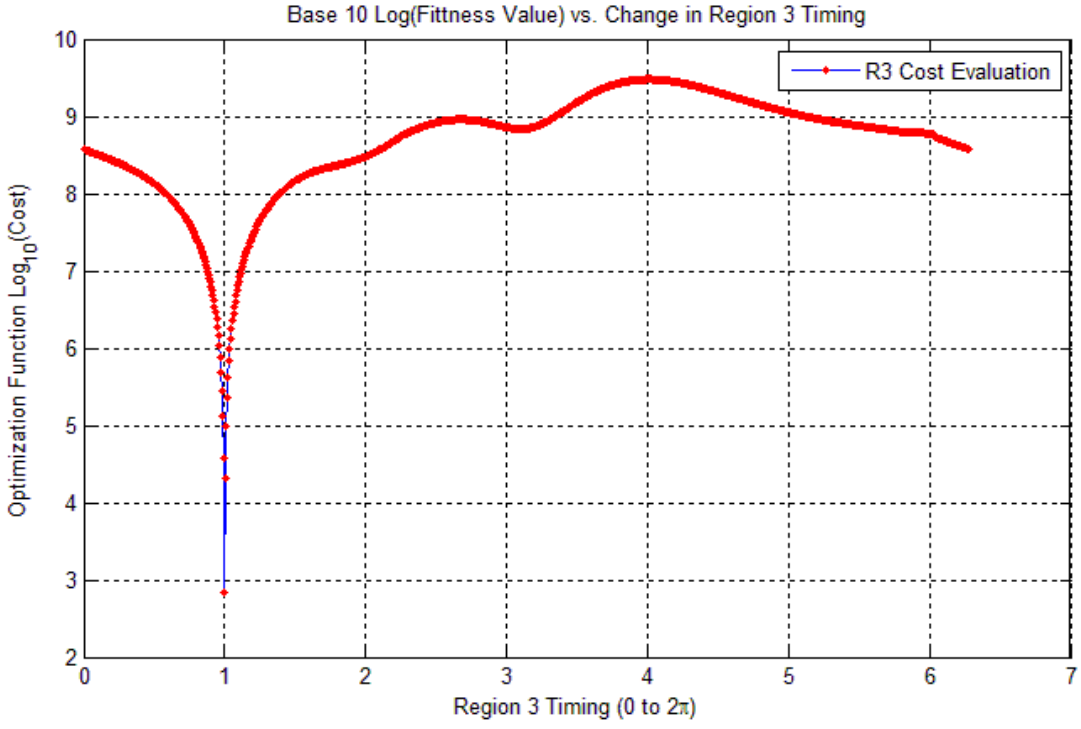

Figure 38: Log_{10}(Cost) Curve for Region 3 Timing Changes Zoomed

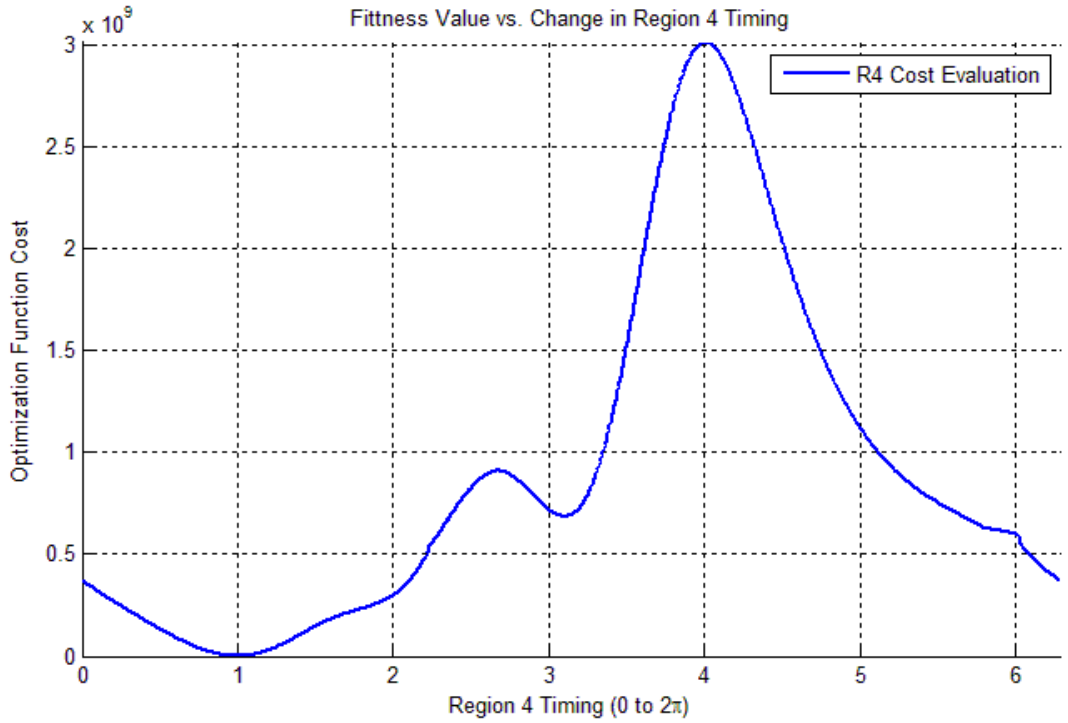

Figure 39: Cost Curve for Region 4 Timing Changes

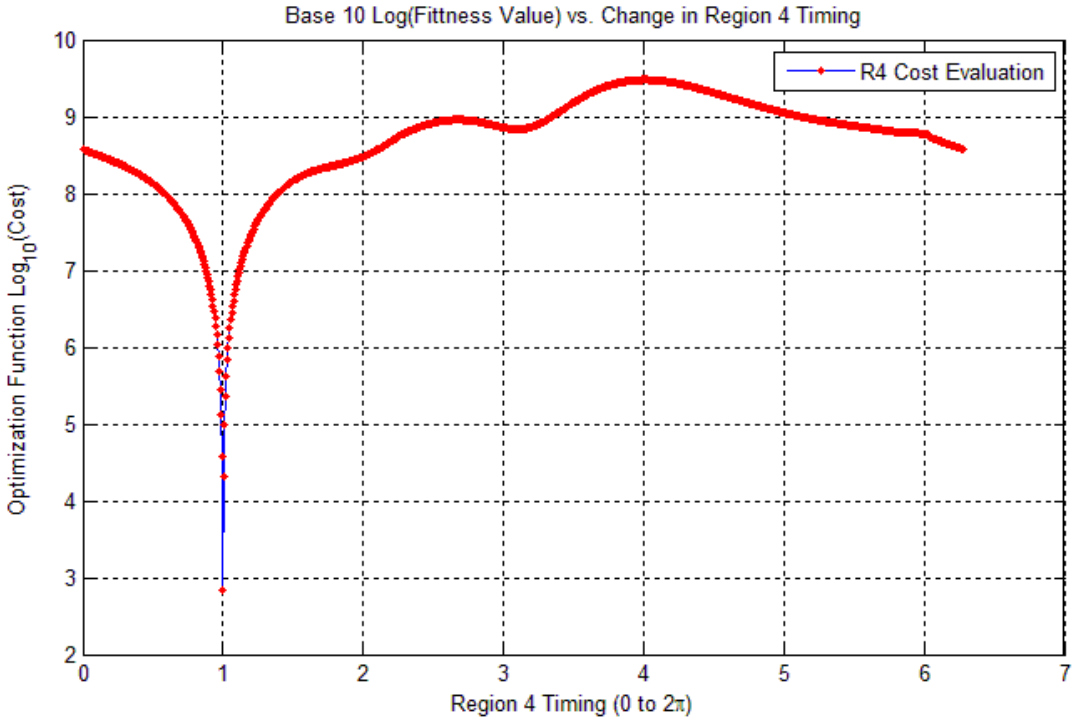

Figure 40: Log_{10}(Cost) Curve for Region 4 Timing Changes Zoomed

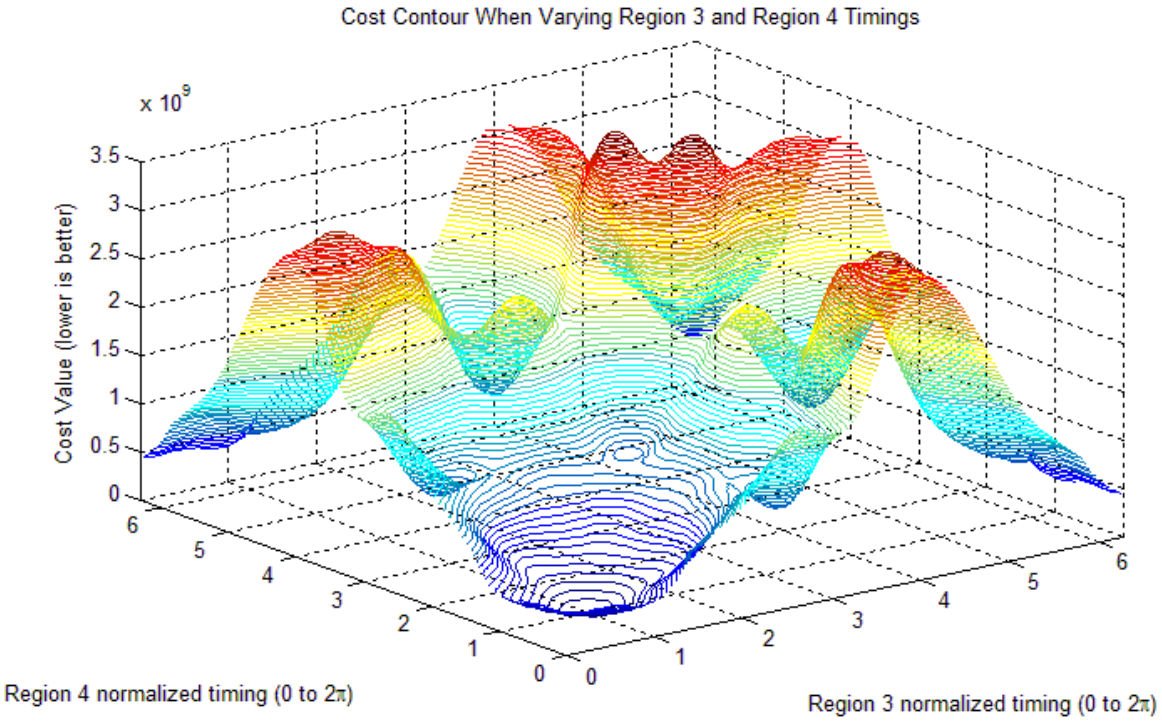

Figure 41: 3D Cost Contour for Timing Changes in Regions 3 and 4

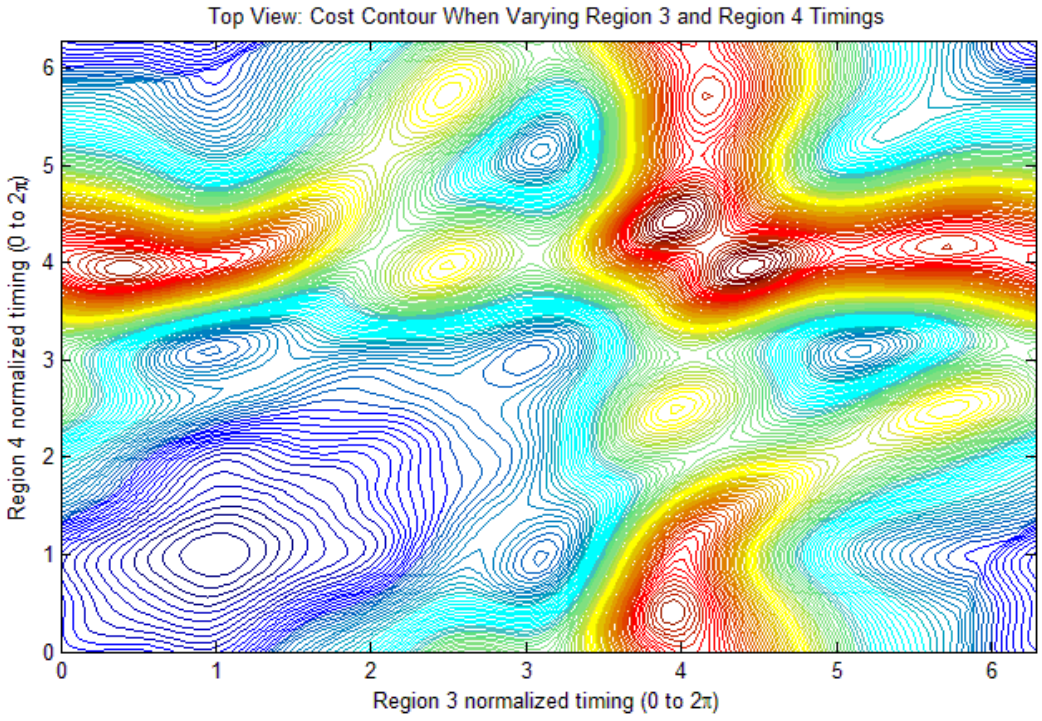

Figure 42: Top View Cost Contour for Timing Changes in Regions 3 and 4

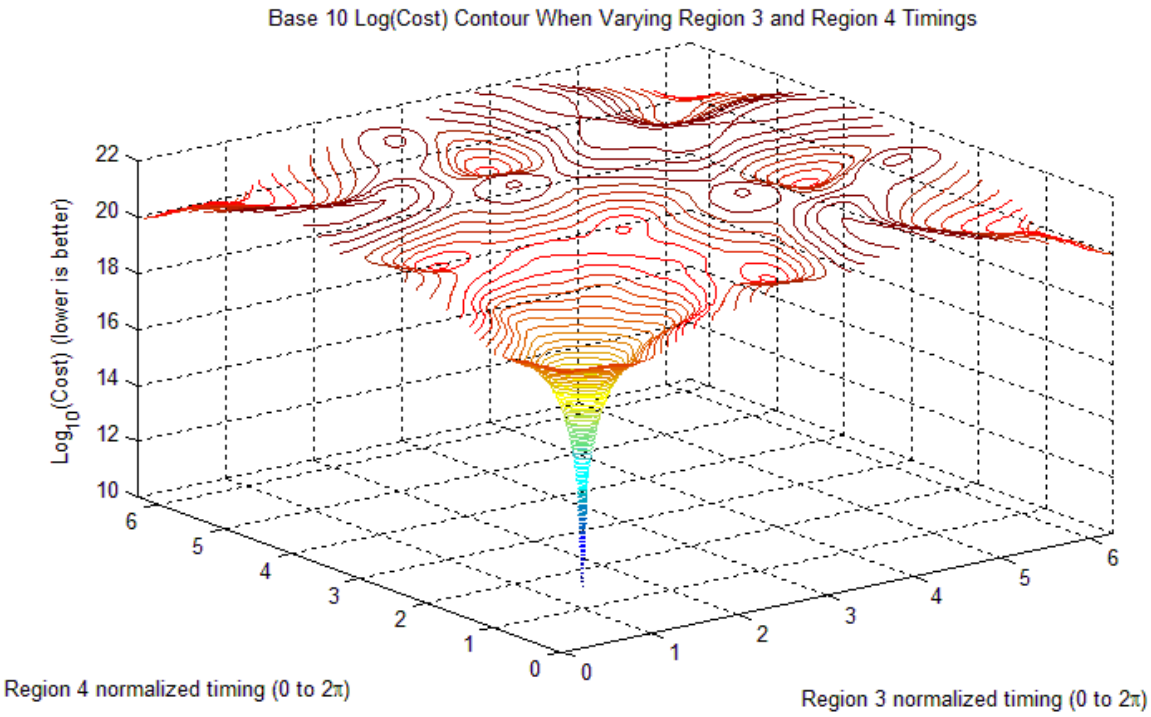

Figure 43: 3D Log_{10}(Cost) Contour for Timing Changes in Regions 3 and 4

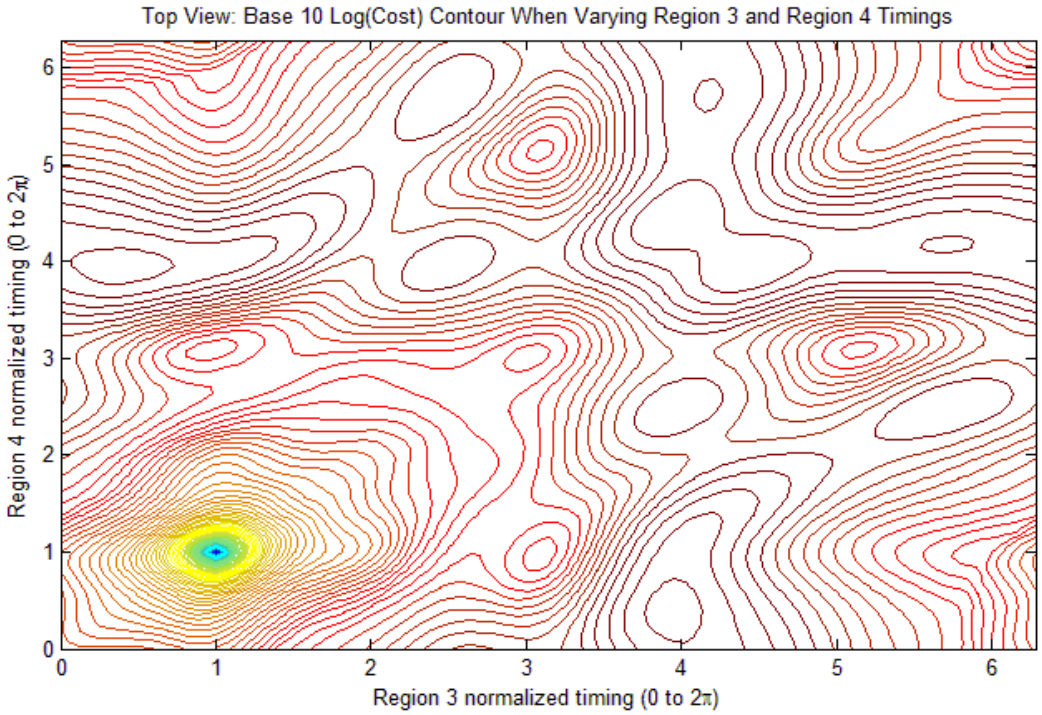

Figure 44: Top View Log_{10}(Cost) Contour for Timing Changes in Regions 3 and 4

5.2.6. Additional GA Investigation

As discussed in section 5.2, GA optimization performs less than desirably in test trials involving more than two parameters (1 reference and 1 controlled parameter). There are multiple possible causes for these results that include: problematic optimization fitness surface geometry, incorrect population size utilization, and incorrect mutation rate utilization. Data for GA cases are introduced in sections 5.2.4 and 5.2.6.1 and are investigated further in sections 6.2.5 and 7.1.

5.2.6.1. Additional GA Trials: Tables

Data for variations in population size and mutation rate are displayed in tables 9 and 10. Further, the average results from 10 runs using each set of initial conditions are plotted in Figure 45 on top of a copy of the contour plot shown in Figure 42 which is generated by having regions 1 and 2 are both equal to 1. While it is noted that region 2 does not always evaluate to 1 in the trials, it is a concession necessary to allow at least a partial visual comparison between various trials.

Table 9: GA Varied Population Size Trials

GA Trial		# Runs	AP_2	t_{Nmin2}	t_{Navg2}	t_{Nmax2}	t_{Nmin3}	t_{Navg3}	t_{Nmax3}	t_{Nmin4}	t_{Navg4}	t_{Nmax4}	$t_{N2,GA,ND}$	$t_{N3,GA,ND}$	$t_{N4,GA,ND}$
ND	**(1,3)**	10	48	0.8682	1.0092	1.1781	0.9143	1.1325	1.7564	0.9219	1.5134	3.2505	1.2907	0.8897	1.2056
ND	**(1,3)**	10	72	0.9695	0.9891	1.0063	0.8529	1.9709	5.5837	0.4157	1.2217	2.3807	1.2907	0.8897	1.2056
ND	**(1,3)**	10	96	0.9158	0.9926	1.0262	0.4249	2.1546	5.7187	0.6918	1.1374	1.8837	1.2907	0.8897	1.2056
ND	**(1,3)**	10	120	0.9986	1.0058	1.0247	0.1917	2.5061	5.9887	0.6673	1.3181	2.8256	1.2907	0.8897	1.2056

GA Trial cont.		EF_{GA}	$EF_{GA,ND}$	%Δ EF	CO_{GA}	$CO_{GA,ND}$	%Δ CO	%Δ F	Fig Ref
ND	**(1,3)**	0.3584	0.3642	-1.5925	1435	1450	-1.0397	-1.00E+02	A-23, A-24
ND	**(1,3)**	0.3318	0.3642	-8.8962	1370	1450	-5.5279	-1.00E+02	A-25, A-26
ND	**(1,3)**	0.3196	0.3642	-12.2460	1326	1450	-8.5907	-1.00E+02	A-27, A-28
ND	**(1,3)**	0.2864	0.3642	-21.3619	1214	1450	-16.2547	-1.00E+02	A-29, A-30

Table 10: GA Varied Mutation Rate Trials

GA Trial		# Runs	AP_2	t_{Nmin2}	t_{Navg2}	t_{Nmax2}	t_{Nmin3}	t_{Navg3}	t_{Nmax3}	t_{Nmin4}	t_{Navg4}	t_{Nmax4}	$t_{N2,GA,ND}$	$t_{N3,GA,ND}$	$t_{N4,GA,ND}$
ND	**(1,3)**	10	0.01	0.8682	0.9916	1.0584	0.7793	1.9411	5.3674	0.7747	1.5993	3.1569	1.2907	0.8897	1.2056
ND	**(1,3)**	10	0.05	0.9695	1.3419	4.1510	0.8575	2.7339	5.5545	0.2102	2.5621	6.1497	1.2907	0.8897	1.2056
ND	**(1,3)**	10	0.10	0.1104	1.0474	4.1832	1.2701	3.1361	5.8000	0.3375	2.1554	5.9350	1.2907	0.8897	1.2056

Table 10: GA Varied Mutation Rate Trials Continued

GA Trial cont.		EF_{GA}	$EF_{GA,ND}$	%Δ EF	CO_{GA}	$CO_{GA,ND}$	%Δ CO	%Δ F	Fig Ref
ND	**(1,3)**	0.0551	0.3642	-84.8710	1400	1450	-3.4427	5.47E+10	A-31, A-32
ND	**(1,3)**	0.0392	0.3642	-89.2367	1099	1450	-24.2173	1.46E+11	A-33, A-34
ND	**(1,3)**	0.0059	0.3642	-98.3800	156	1450	-89.2558	1.55E+11	A-35, A-36

5.2.6.2. GA Investigation: Contour Plot Representations

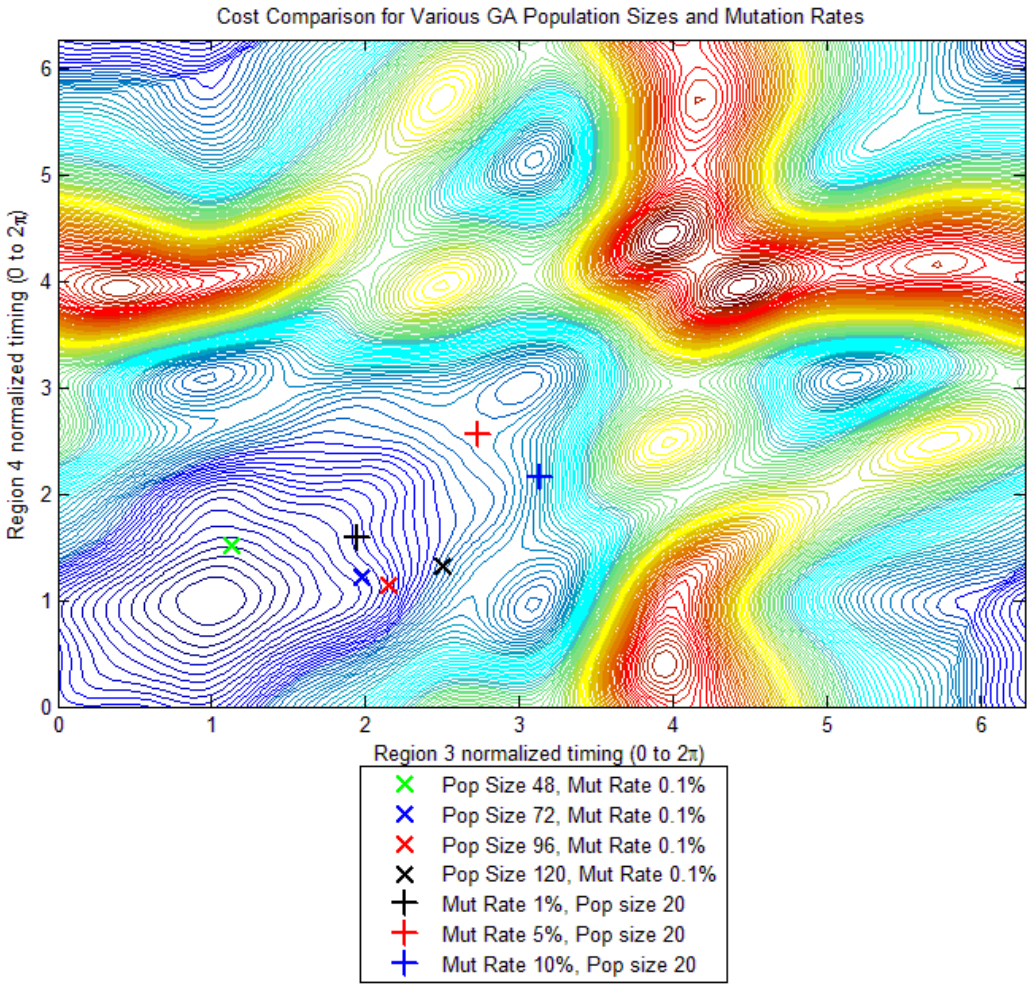

Figure 45: Overall GA Parameter Variation Cost Comparison, Region 2 Values Truncated to 1

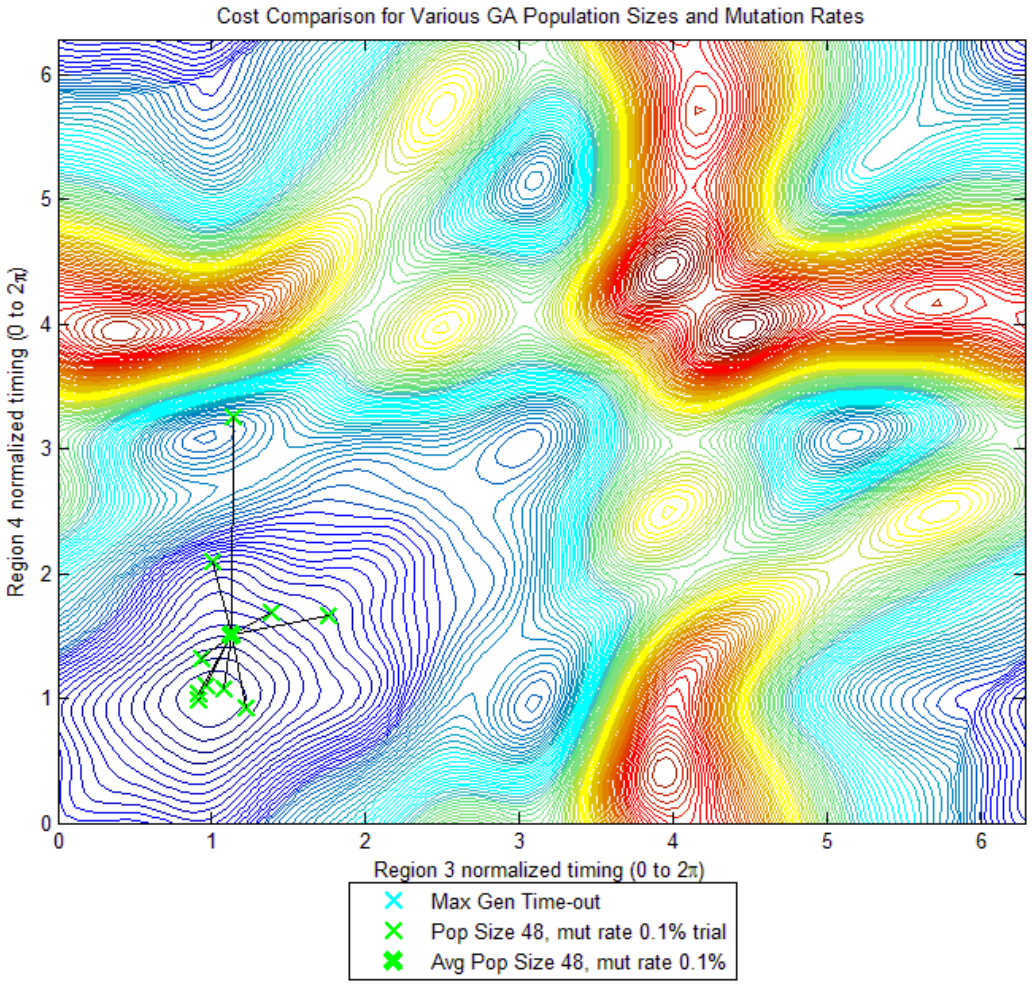

Figure 46: GA Population Size = 48 Individual Run Cost Comparison, Region 2 Values Truncated to 1 for Comparison

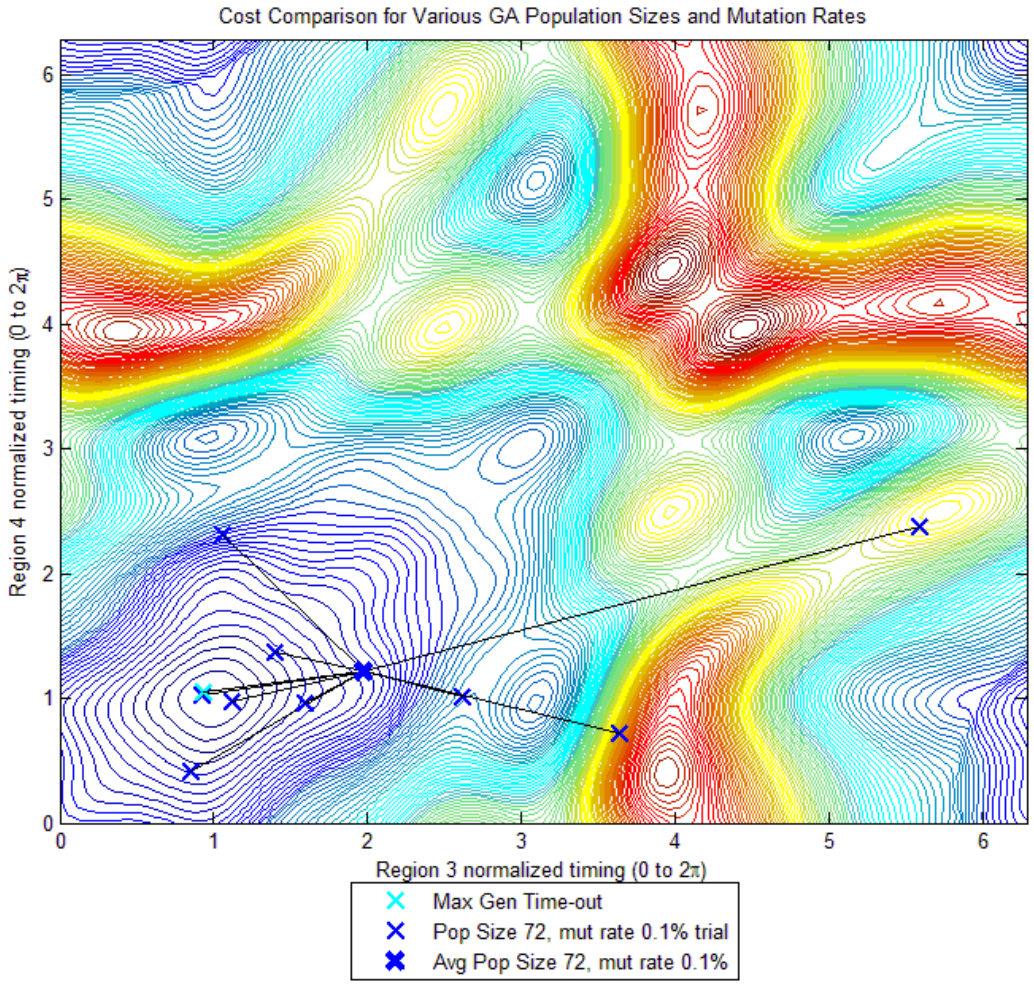

Figure 47: GA Population Size = 72 Individual Run Cost Comparison, Region 2 Values Truncated to 1 for Comparison

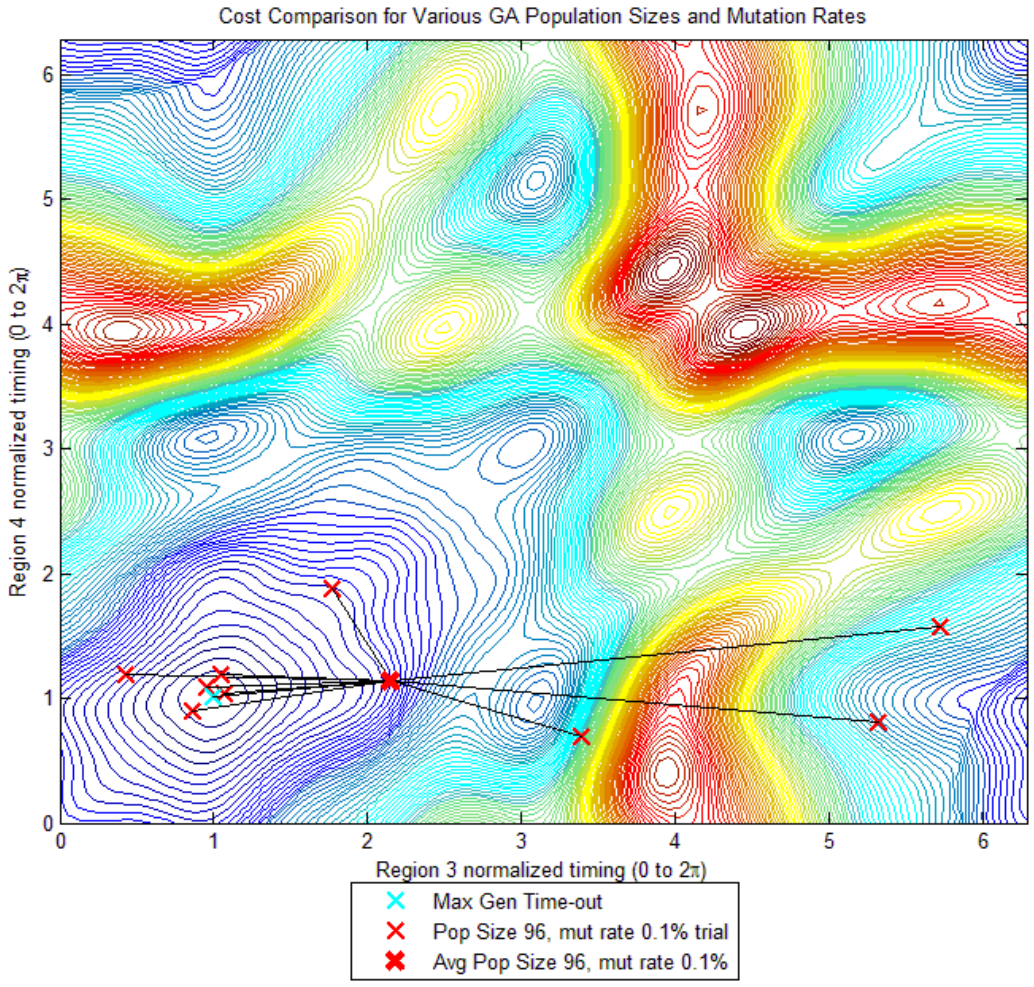

Figure 48: GA Population Size = 96 Individual Run Cost Comparison, Region 2 Values Truncated to 1 for Comparison

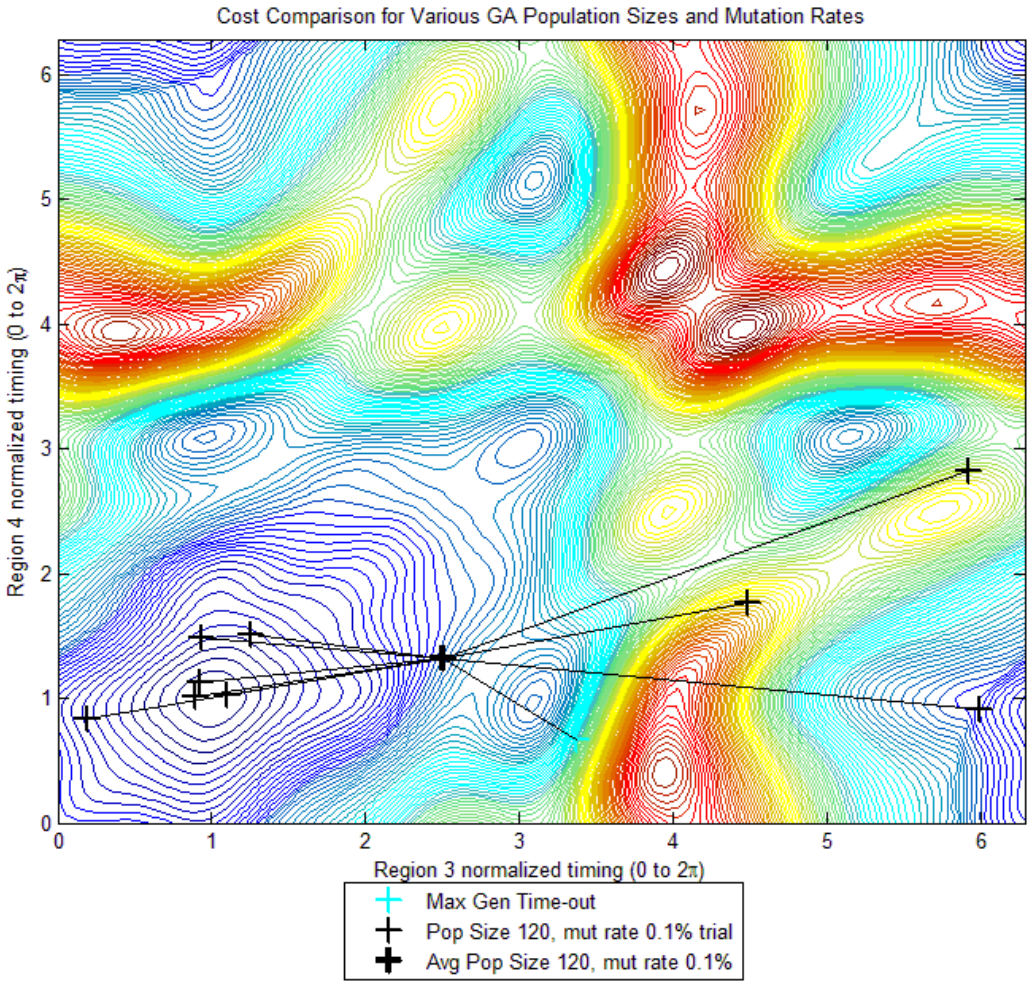

Figure 49: GA Population Size = 120 Individual Run Cost Comparison, Region 2 Values Truncated to 1 for Comparison

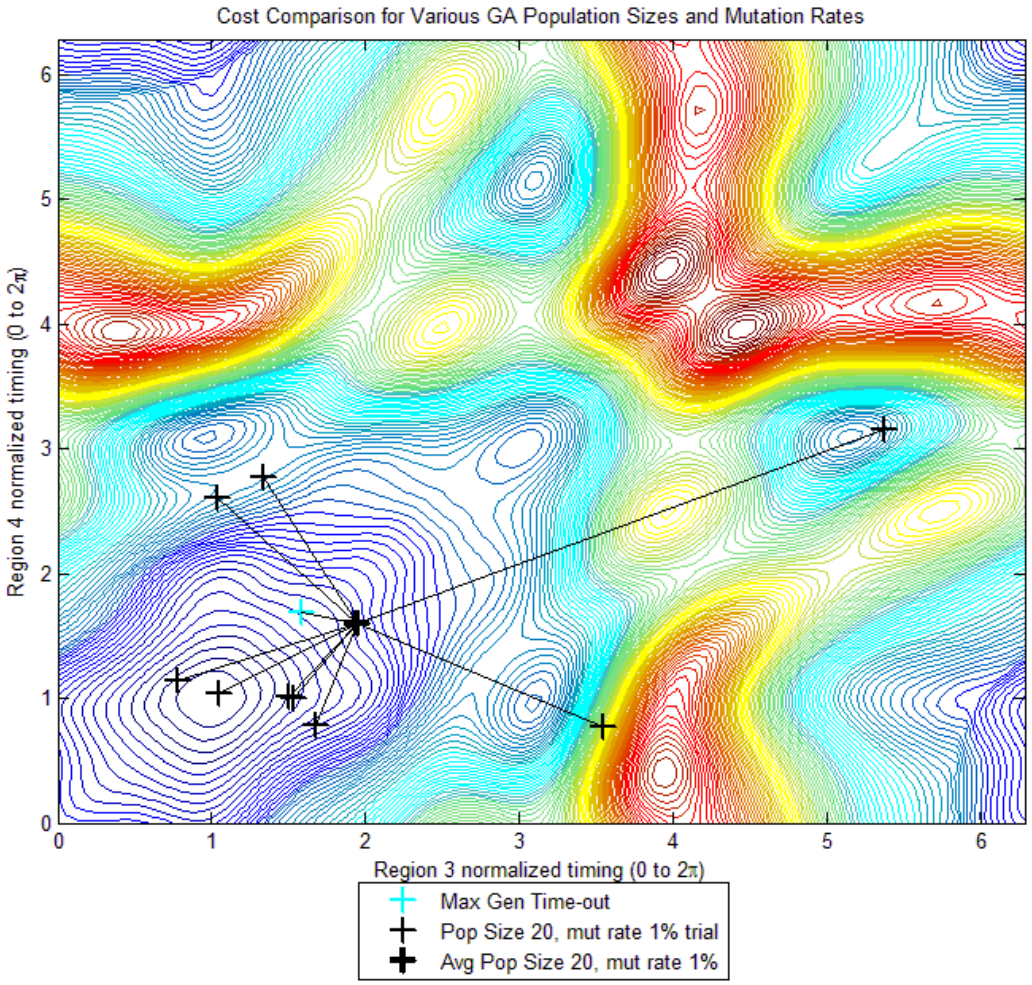

Figure 50: GA Mutation Rate = 1% Individual Run Cost Comparison, Region 2 Values Truncated to 1 for Comparison

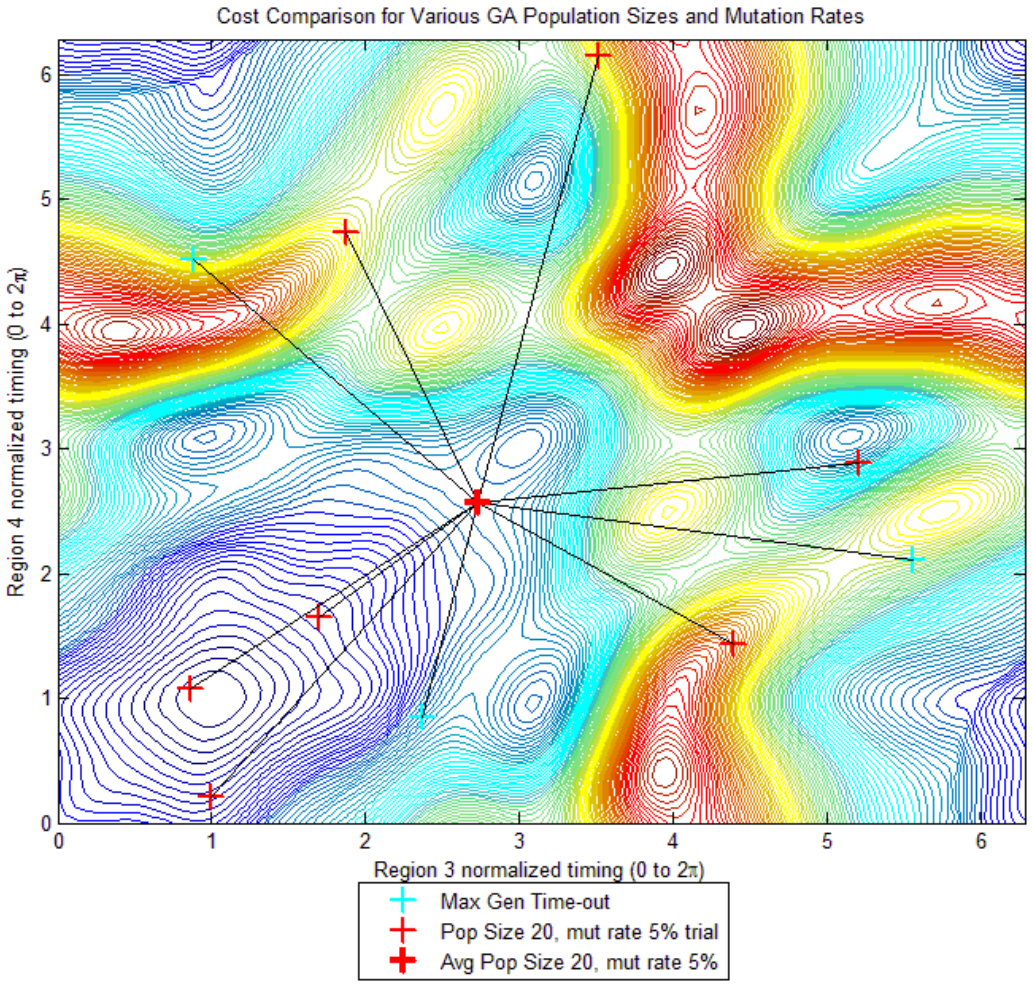

Figure 51: GA Mutation Rate = 5% Individual Run Cost Comparison, Region 2 Values Truncated to 1 for Comparison

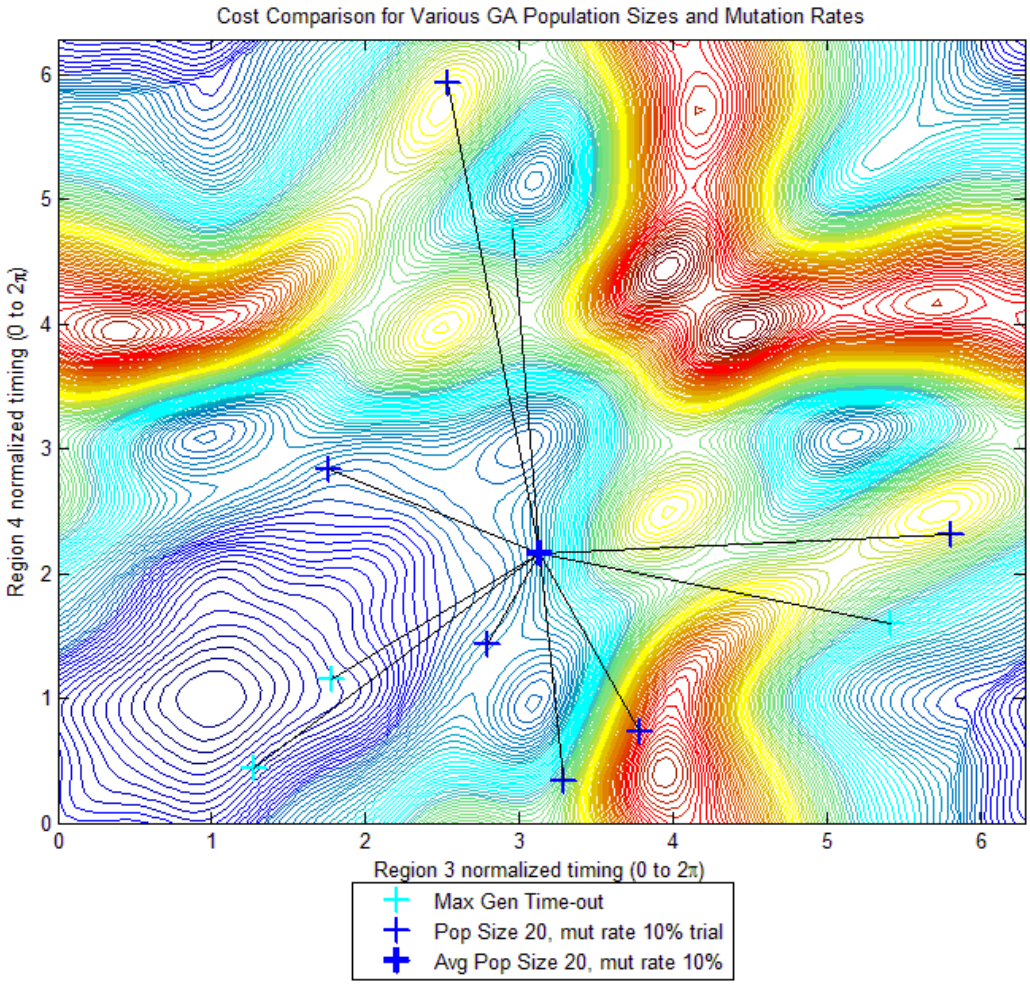

Figure 52: GA Mutation Rate = 10% Individual Run Cost Comparison, Region 2 Values Truncated to 1 for Comparison

5.2.7. Combined GA and Exhaustive Search Timing Comparisons

Sections 5.2.1 and 5.2.2 provide tabular data for the results generated by both GA and exhaustive searches respectively in trying to minimize ventricular wall acceleration for the initial conditions provided for the trial. Additionally, section 5.2.6.1 shows tabular data from investigating the impact of population size and mutation rate GA search parameters on the ability

of the GA to converge to a more or less fit solution. The average regional timing results from the GA trials and the returned timings from the exhaustive search are placed in box and whisker plots in sections 5.2.7.1-3 broken into 2 parameter trials, 4 parameter trials, and additional GA trials respectively to provide a visual representation of the results obtained.

5.2.7.1. 2 Parameter Timing Box and Whisker Plot

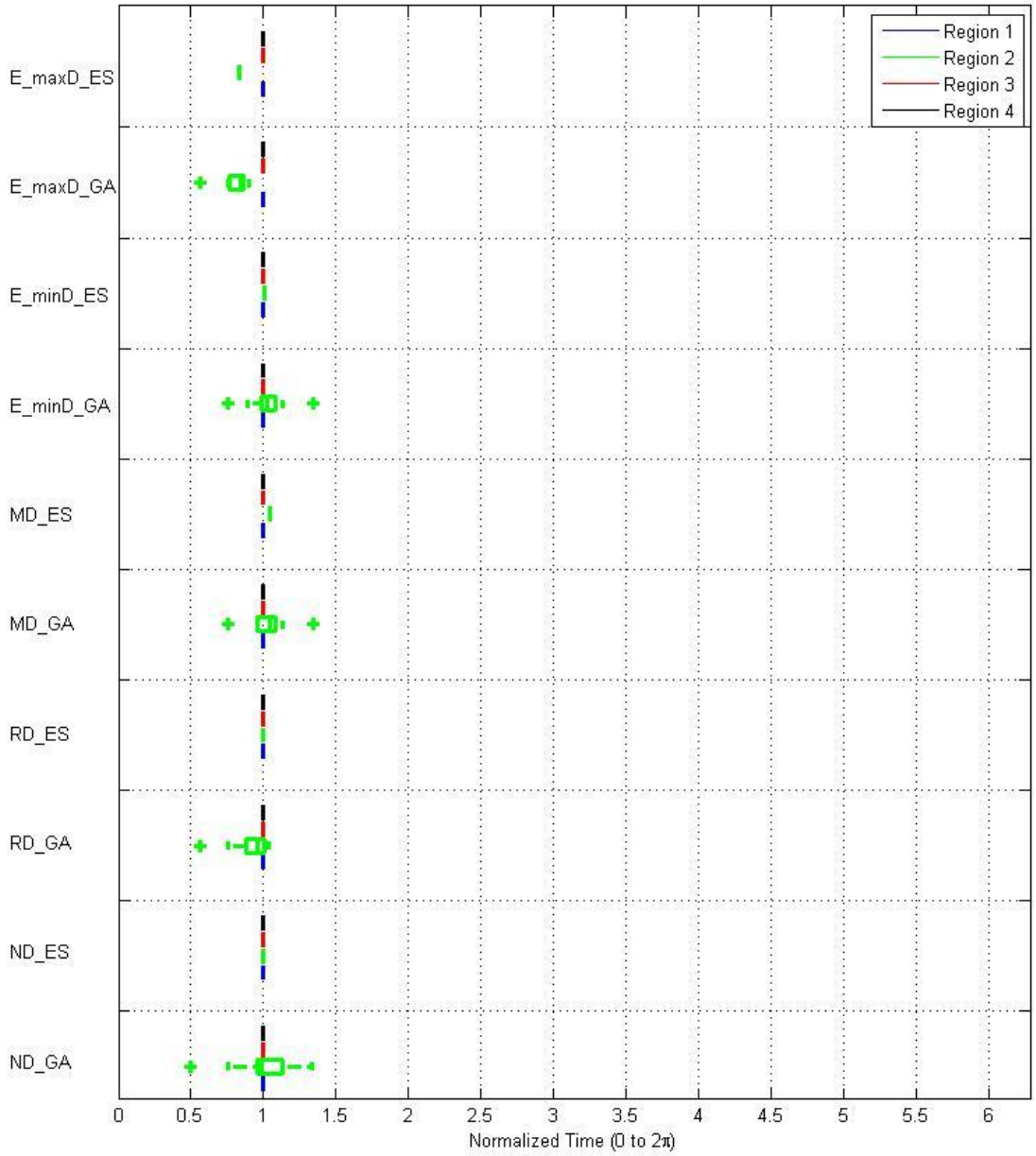

Figure 53: 2 Parameter Box and Whisker Plot Multiple Trials: GA vs. Exhaustive Search

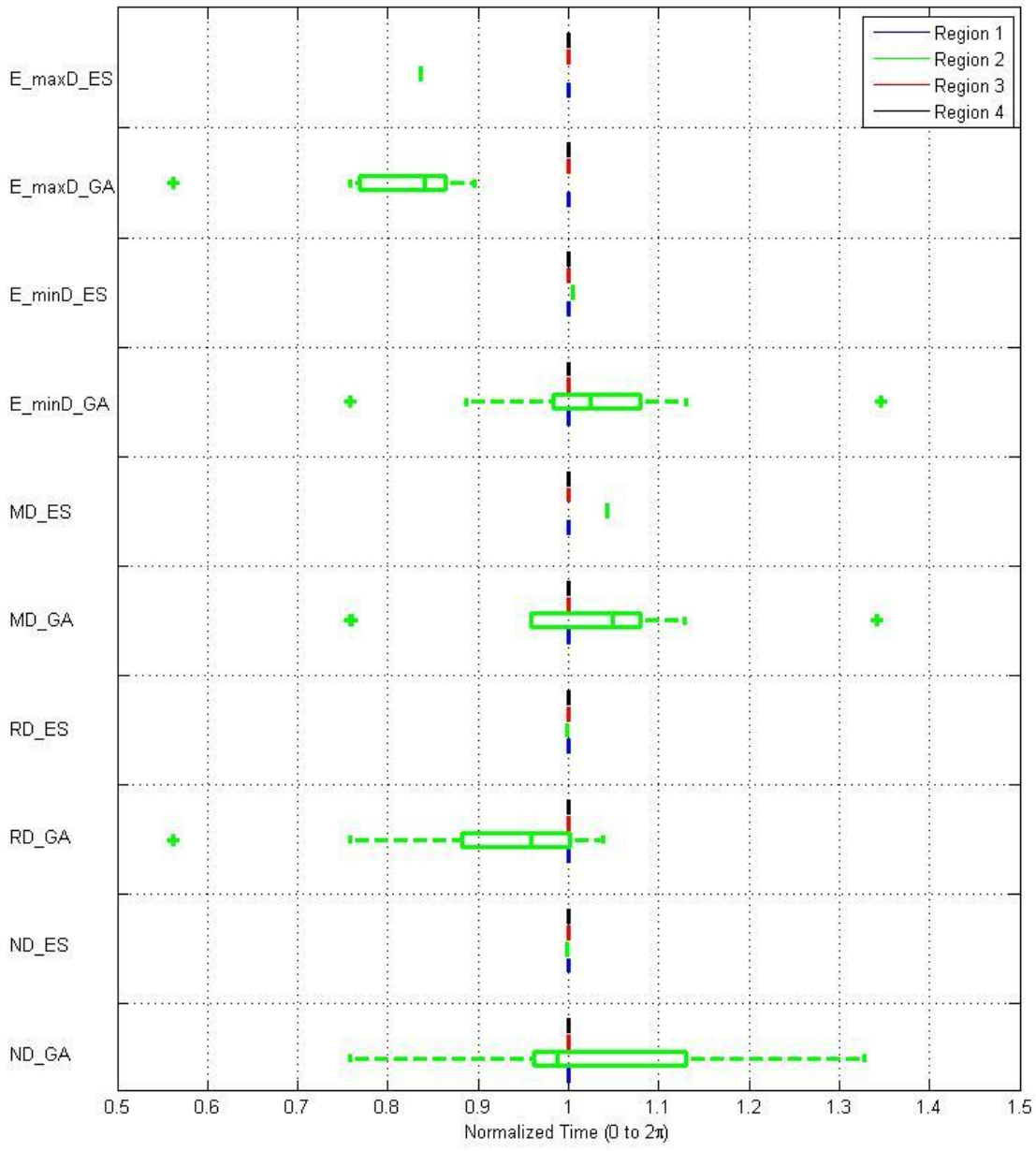

Figure 54: 2 Parameter Box and Whisker Plot Multiple Trials: GA vs. Exhaustive Search Zoomed

5.2.7.2. 4 Parameter Timing Box and Whisker Plot

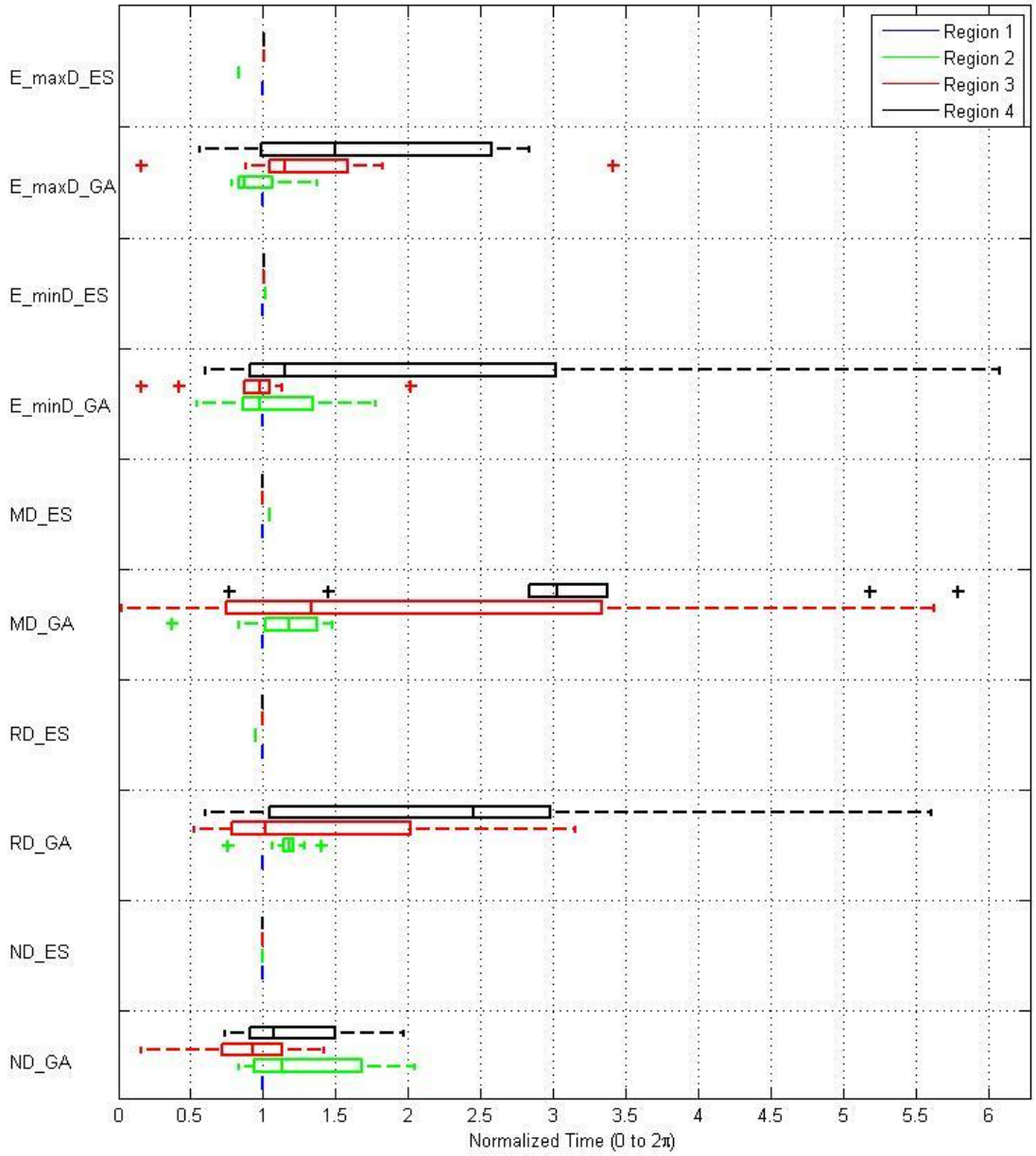

Figure 55: 4 Parameter Box and Whisker Plot Multiple Trials: GA vs. Exhaustive Search

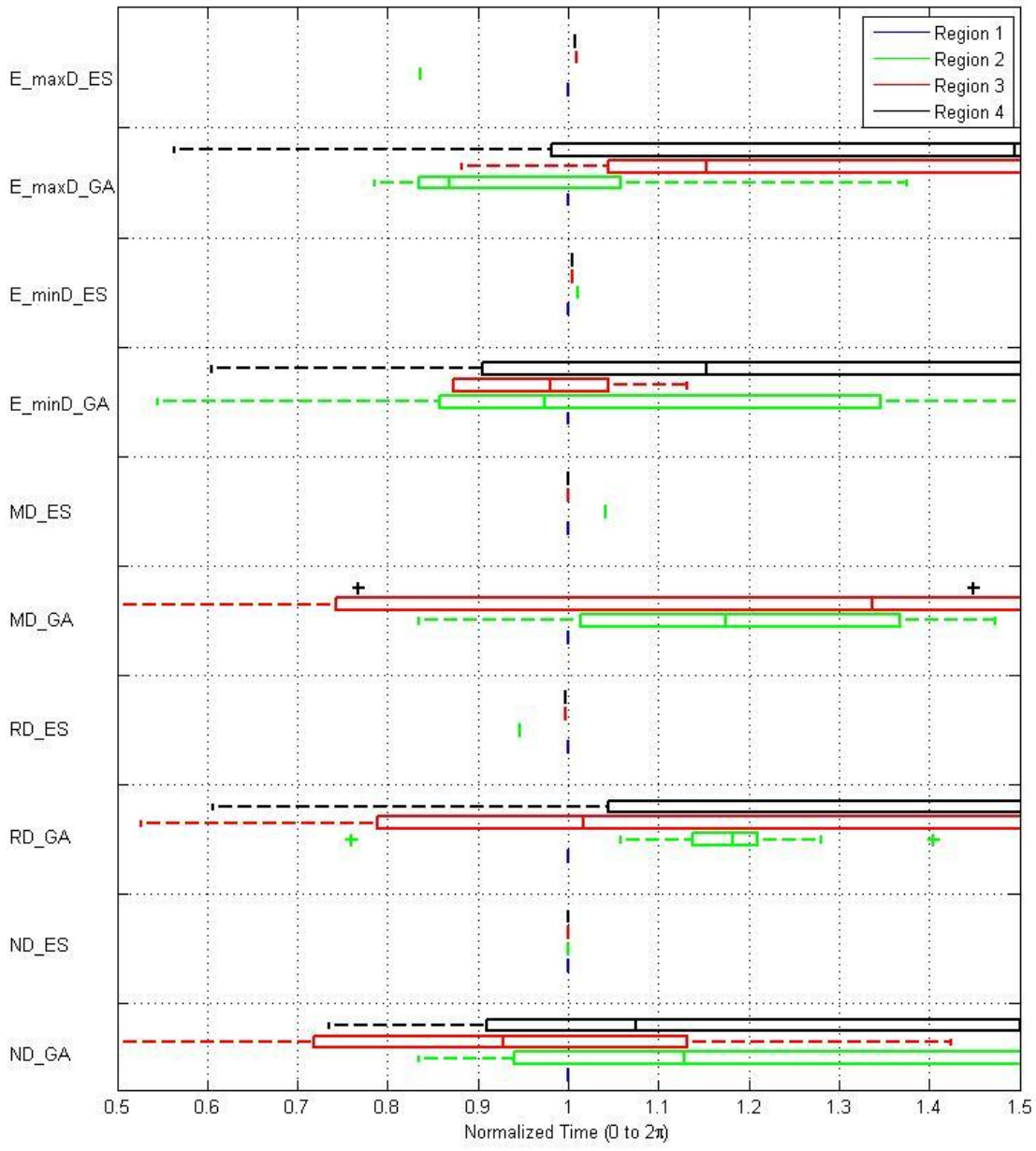

Figure 56: 4 Parameter Box and Whisker Plot Multiple Trials: GA vs. Exhaustive Search Zoomed

5.2.7.3. Varied GA Parameters Box and Whisker Plot

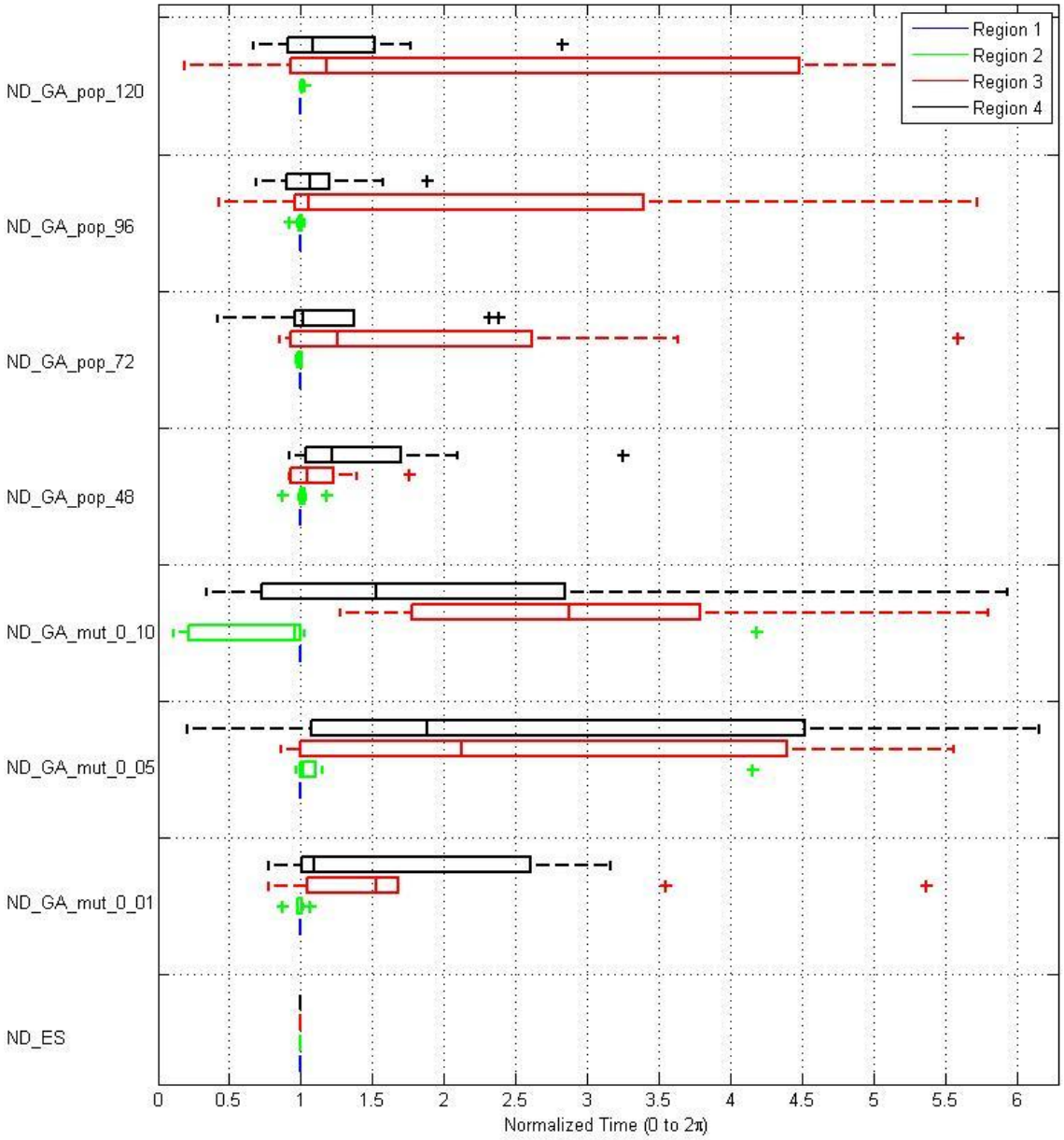

Figure 57: 4 Parameter Box and Whisker Plot Multiple Trials: GA Variations

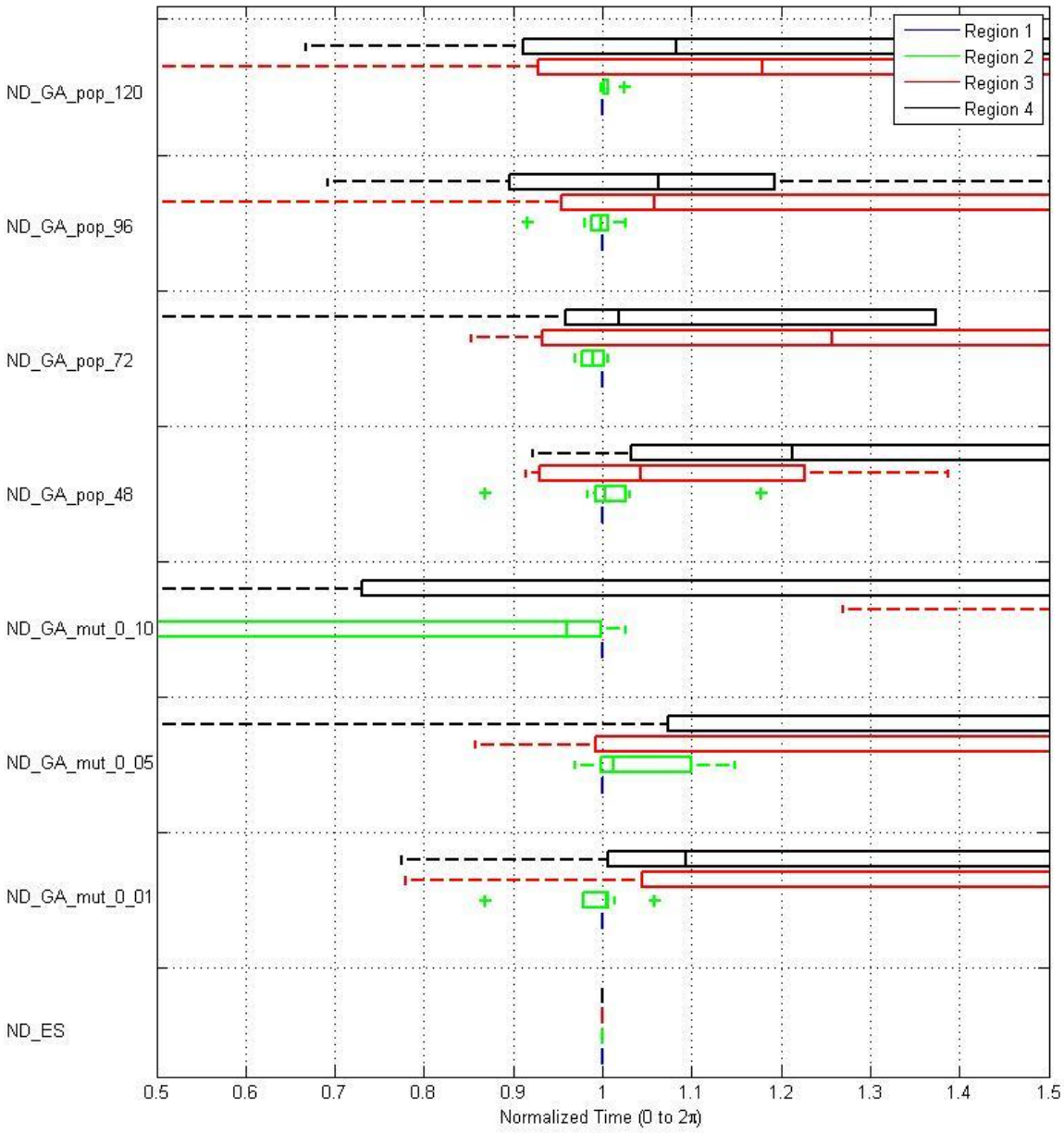

Figure 58: 4 Parameter Box and Whisker Plot Multiple Trials: GA Variations Zoomed

6. DISCUSSION

6.1. Interpreting the Walking Sinusoid Model GA Search Data

In the case of the walking sinusoids model, the GA was able to converge with one and two searched parameters by averaging the results of 10 trials. Due to the nature of stochastically based optimization algorithms like GAs, the error seen in individual final outputs is not surprising. With the simplistic nature of the model, the time required to execute a GA based search is likely longer that what might be required by other methods such as exhaustive search, even at a relatively fine grid-like search of the solution space. Though another method may be more time efficient in generating a solution, the purpose was to validate the GA algorithm designed prior to use with more computationally intensive models. Regardless, the GA succeeded in its task even under noisy conditions as shown in Table 1 and in Appendix F Figures.

6.2. Discussion and Analysis of CD Model Testing

Due to a model discrepancy found late in the data collection process and the large number of trial cases already performed, select trial cases based on the model improvements were performed and analyzed to inspect the impact on previously collected results. These new trials mirror one of the previous 4 parameter control trials and one additional random sample from the previous 4 parameter dyssynchrony trials to provide a comparison of what impact the model setup may have on the data already collected in the previous model configuration. Additional information was also gathered for these trials to draw more insightful conclusions and to provide a basis for determining future directions. Such additional information includes PV loop plots, Wiggers diagrams, and regional work/power plots for these select trials.

6.2.1. CD Model Validation

As mentioned in section 6.2, after multiple test trials, it was found that the CD model being used contains some discrepancies in some of its initial conditions that contributed to volumes outside of physiological norms. Attempts were made to work with the model designers to remedy the defect and the corresponding values were brought within acceptable ranges; however, it was not possible to create a situation in which all the nominally accepted values were obtained simultaneously which can likely be attributed as a shortfall of using a simple model for a very complex system. After varying model initial conditions to elicit more physiologic values for pressure and volume, the CD model with varied initial conditions was used with the 4 parameter exhaustive search to see if those model conditions affected optimization timings.

From this, two trials were selected to be re-run using the new model conditions: the control, no dyssynchrony trial and a random trial from the set of dyssynchrony trials, the maximum elastance dyssynchrony trial. The new initial conditions involve changing the value of "volume" in the dyss1.m file from 10 to 30, and the default minimum elastance was decreased to 0.5 from 1 in Scripted_Initialization.m. In addition, Line 11 of dyss1.m was changed from …ode23s(…6 100 volume…) to …ode23s(… pressure 100 volume). While these changes do not result in the expected "normal" physiological values for ventricular volume, pressure, stroke volume, ejection fraction, or cardiac output, the values do better fall within physiological ranges.

The no dyssynchrony trial converges to the same solution with both the new and old CD model initial conditions as shown in Table 2. Thus, it can be concluded that the model changes do not impact the ability for a solution to be realized through optimization.

Also shown in Table 2, the dyssynchrony trial $E_{max}D$ indicates approximately a 3.2 ms, overall shift in optimal timings, but the distance between the optimized regional times are largely

unaffected. This can be interpreted to indicate that while the model's initial conditions can drastically impact modeled ventricular volumes, pressures, stroke volumes, and cardiac outputs, the impact on wall acceleration values and the corresponding wall acceleration optimizations is influenced to a much lesser extent. From this, it can be reasoned that while the absolute timing values and percentage improvements may differ, the overall conclusions drawn from these trials can largely be translated to the pre-existing trials using the old model initial conditions.

The results of this testing show that while the acceleration waveforms are altered, the timings returned as optimal remain largely unchanged from those using the previous version of the model and initial. From this it was concluded that while the acceleration waveforms may be inaccurate representations on the majority of the test cases, the results and conclusions derived from the earlier test cases are still valid.

6.2.2. Cardiac Pressure-Volume Characteristics

Figures 17 and 18 respectively show the PV loop characteristics for the no dyssynchrony and maximum elastance dyssynchrony trials involving the new CD model initial conditions. The figures show both the exhaustive search optimized timings as well as a no adjustment trial in which timings for all 4 regions are forced to be equal. (The no adjustment trial simulates a pacemaker setting regional timings without a feedback parameter for self adjustment.) In the case of no dyssynchrony (Figure 17) it is shown that the exhaustive search is capable of matching the performance of a pacemaker forcing simultaneous activation of all 4 regions. By contrast, Figure 18 shows in the case of maximum elastance dyssynchrony both end-diastolic and end-systolic volumes are reduced without a commensurate increase in pressure. Rather, a subtle shift in ventricular pressure in the timing optimized condition is seen when compared to the trial when timings are forced to be equal. While this translates into a higher EF, CO is largely

unchanged. A reduction of both end-diastolic and end-systolic volume is a desirable characteristic since the increased preload and afterload associated with higher end-diastolic and end-systolic volumes can lead to ventricular remodeling such as hypertrophy [33].

6.2.3. Acceleration and Timing Characteristics

References to individual figures are given in tables 2 through 10. For example, Figure 21 and 25 correspond to the CD model under these three conditions: new initial conditions, no dyssynchrony and 4 parameter optimization. Figure 21 shows the acceleration waveforms as would be measured on the ventricular walls for each of the 4 regions computed by the model. In 2 parameter mode, 3 of the regions are tied together and set to the control timing while the 4^{th} region is allowed to be optimized by the named algorithm. The acceleration waveforms are color coded to their respective regions and a legend is provided with each plot. Some trials also show the timing values used to generate the acceleration waveforms on the acceleration plots, like seen in Figure 21. As the regions are on either side of a central chamber as shown in Figure 15, acceleration direction was normalized such that movement toward the center indicates a negative acceleration, and movement away from the center indicates a positive acceleration regardless of the region's location. In this context, direct overlay of all regional accelerations is the most desired outcome.

Figure 25 indicates the times at which each region was commanded to "fire" (begin contracting) with respect to the overall heart timing signal. These values range from 0 to 2π with a value of 1 being the default time of the fixed region(s). The color coded lines indicate the solution(s) returned by the search algorithm utilized for the controlled region(s). A legend is provided with each graph to aid in identification of regional results and performance. In the case of GA sets where multiple individual attempts were made, there are multiple, thin, tick marks

indicating the individual GA results returned, and a thicker tick mark indicating the average of all individual GA runs for a specific set of starting conditions. In the case of exhaustive search, and no adjustment runs, there are grey, vertical bars indicating the range of possible solutions for controlled region(s) to show the final level of granularity when performing the exhaustive search since the ability to draw meaningful conclusions from the exhaustive search is directly proportional to how fine the search grid was when the search terminated.

Figures 23 and 24 correspond to maximum elastance dyssynchrony using exhaustive search and no optimization respectively. As can be seen in these figures, the exhaustive search optimized trial has better correlation during systole with some sacrifices made during diastole when compared to the non-optimized, linked, firing times.

6.2.4. Regional Work and Power Plots

Figures 29-32 show regional work plots for no dyssynchrony and maximum elastance dyssynchrony under both timing optimized and non-optimized conditions respectively. In the case of no dyssynchrony, Figures 29 and 30, under both timing optimized and non-optimized conditions, no work variations are observed between regions. However, in the case of maximum elastance dyssynchrony, Figures 31 and 32, it can be seen that there is approximately a 50% increase in work performed in Region 2 (the region with varied maximum elastance) for the timing optimized condition when compared to the non-optimized control with approximately 15% decreases in work performed by the other three regions resulting in a much more balanced work distribution. This redistribution of work load likely explains the lack of significant CO change observed in the other dyssynchrony trials.

In similar fashion, Figures 33-36 show regional instantaneous power generation for no dyssynchrony and maximum elastance dyssynchrony under both timing optimized and non-

optimized conditions respectively. Again, there is no observable difference between optimized and non-optimized conditions in the no dyssynchrony trial, Figures 33 and 34. However, in the case of maximum elastance dyssynchrony, the maximum power generated by region 2 increased by approximately 50% while the remaining regions decreased by approximately 8% each resulting in a much stronger correlation between peak power generation of region 2 and the remaining regions. [27] and [31] link variations in regional workload to cardiac remodeling that is ultimately detrimental. Similarly, it would be interesting to investigate if balancing those regional workloads elicits more beneficial remodeling.

6.2.5. Interpreting the CD Model GA Search Data

In the case of the CD Model, the GA performed unsatisfactorily in the 4 parameter model search by investigation of the ΔF% values. However, by the same ΔF% value, GA use on the 2 parameter model is more promising with only +/- 7% deviations from the exhaustive search solutions involving the same conditions as shown in Table 3. Visual comparison of these timing results can be observed in Figures 53 – 56 where each colored plot represents the range of individual solutions for the specified region. The box represents the middle 50% of data with the outer edges marking the upper and lower quartiles (25^{th} and 75^{th} percentiles), and the center mark in the box representing the mean (50^{th} percentile). Cross marks designate outliers falling outside +/- 2.7 standard deviations from the mean.

With the exception of the RD, 2 parameter trial, these solutions still do not reach the level of accuracy and cost reduction achieved by the exhaustive search. This singular improvement can largely be attested to the complexity of the search space as shown in Figures 41 – 44, and the granularity of the exhaustive search used in the 2 parameter conditions as discussed in section 3.2. In contrast however, the 2 parameter GA trial solutions prove to be much closer in terms of

change in cost values (ΔF%) than their 4 parameter trial counterparts outlined by comparing Tables 3 and 4. There are a number of possible reasons for this. The largest suspected culprit for the diminishing performance with increased degrees of freedom is premature convergence of the population. Although the ultimate cause is unknown, contributing factors may include: an undersized population pool, overly fit individuals relative to the rest of the population early in the GA process dominating mate selection, or too small a mutation rate to name a few. Variations in population size and mutation rate are investigated individually. However, the complex interactions of GA parameters and the implementation of gene manipulating operations do not allow for simple conclusions to be drawn on the individual impact of either parameter.

In short, additional tuning is still required to optimize the GA for solving the problem of reducing acceleration CD by altered regional timings, especially with larger degrees of freedom. Brute force methods such as forcing excessively large population sizes or increasing the maximum number of generations can be used in an attempt to resolve the aforementioned issues of GA, but at a great cost of computational time in each case. Each trial consists of a set of 10 GA runs using the sets of parameters stated previously (section 6.2.2) and was executed using the parallel computing toolbox in MATLAB in a six core environment; these trials averaged about 24 hours for completion. Conceivably, in single core operation, the same task would take up to 6 times as long (almost a week) to converge to a solution—a solution that is not yet correct to any level of implementable usefulness (see Appendix A for examples, trial references can be found in Tables 3, 4, 9, and 10).

To investigate possible contributions from incorrect population sizes and mutation rates, these parameters have been investigated individually as shown in Tables 9 and 10. The averaged results of 10 runs in each trial are placed in Figure 45 for a visual representation. Timing results

for these trials are also compared with those of exhaustive search in Figures 57 and 58 utilizing box and whisker plots discussed previously.

Figures 45 – 52 are visual aids to help understand GA convergence patterns under varied population size and mutation rates, but are not specific to individual GA trial solutions. Since these plots can only show variations in two parameters (timings for regions 3 and 4) at a time. The other values (timings for regions 1 and 2) must remain fixed and have no way of being adjusted from one run to the next. For example, a similar comparison can be made between variations in regions 2 and 3, regions 3 and 4, or regions 2 and 4. However, since region 2 is the most constant in terms of convergence seen in Tables 9 and 10, it was deemed most useful to show the fitness surface from varying regions 3 and 4.

While results from these additional trials (Figures 46 – 52) show variations in the individual solution timing values, it is unclear that modifications to any one parameter provides the desired GA response of all runs converging near the ideal solution as seen in the variation of individual solution runs in Figures 46 – 52.

It is suspected that it takes a combination of parameter changes to allow for consistent convergence using GA optimization and even then, the computational cost may make GA optimization in its native form infeasible for real application. These GA parameter variations could be an avenue for future investigation.

In addition to the potential shortfalls mentioned, the method of un-weighted means for averaging the individual GA runs appears to cause problems with inaccurate GA results, possibly giving the GA less credit than it deserves. As previously discussed, a subset of GA trials is represented in Figures 46 – 52 that show locations of individual solutions with respect to generated values for regions 3 and 4. As can be seen, while some runs converge to regions

bordering the optimal solution when looking at the overall fitness curve, other runs get stuck in local minimums and are unable to escape (unable to remove themselves from a local minimum) prior to population convergence. The averages in each trial, indicated by the bold marks, represent the mean of all compiled GA run solutions within the specified trial. A number of these individual runs in multiple trials converge near the ideal values; however, in averaging all trials, the results from runs that get stuck in either local minima or on the far side of the graphed solution range. The means are computed using standard averaging techniques; however, since the region being investigated is periodic, this method of averaging can unjustly skew results in the event of irregular convergence around the ideal solution. A better solution may be selecting the minimum solution from the set of trials, selecting a solution based on cluster recognition, or obtaining a solution from modified averaging to take advantage of the periodic nature of the search space. While just taking the minimum can impact noise immunity, this method can be further adjusted if noise becomes a problem once measurements are used instead of models.

One additional cause for the varied GA convergence could be the search population was not seeded with a test case that forces all regions to fire simultaneously as a starting point for the search. While this test case could focus the search near that area of the search space if it is a relatively fit solution, introducing it too early in the GA search process could negatively influence the mating probabilities computed and lead to premature population convergence. It was decided that investigating the GA performance in an unguided search is more beneficial since fitness surfaces can conceivably vary greatly depending on the model and initial conditions used. If the GA can converge without an overly specific set of starting conditions itself, it strengthens the argument that GAs can be robust tools for finding solutions to complex problems.

The deviations in CD Model behavior could also potentially lead to less than desirable GA performance for the 4 parameter optimization cases. Although it was found that changes in initial conditions of the model do not significantly impact the optimal condition overall, it may impact the fitness surface in a way that is even less favorable to GA optimization. Further discussion of findings involving CD Model behavior can be found in section 6.2.1.

6.2.6. Interpreting the Exhaustive Search Data

Exhaustive searching of the solution space yields rather exciting results. As can be seen in Tables 3, 4, 5, and 6 along with Appendix B Figures, the Exhaustive Search yields better solutions than all 4 parameter GA solutions and meets or exceeds the "no adjustment" solutions of firing all regions at the same time. The only time firing all regions at the same time out performed exhaustive search solutions is in the case of no dyssynchrony which can be attributed to discretization errors when sampling and testing the search space. This error could be minimized further and nearly eliminated if the exhaustive search is allowed to narrow its search regions further, at the cost of computational time. As an aside, while a normal heart largely contracts from the bottom up, the simplest solution to CD is firing all regions simultaneously. The default case for comparison, "no adjustment", is used to generate baseline data matching the simplest solution, not normal physiology. This baseline is used in the ultimate determination of whether or not timing adjustments can beat baseline CD correction. All being considered, since the main goal of both the exhaustive search and GA search was to test whether changes in regional timing can improve ventricular wall acceleration dyssynchrony, the results for the no dyssynchrony case were treated as a control in verifying the algorithms performed as expected. In conclusion, exhaustive search outperforms both GA and "no adjustment" cases with the singular exception to the discretization errors previously discussed.

7. CONCLUSIONS

7.1. GA Conclusions

Overall, the GA algorithm and procedure outlined in this thesis takes a step-by-step approach in investigating the use of ventricular wall accelerations as a measure to adjust regional firing times in the pursuit of reducing acceleration measured cardiac dyssynchrony. This step-by-step procedure resulted from the complexity of implementing a GA, and was put into practice by giving a GA tutorial and slowly building up a GA using simple and intuitive models. This method allowed for absolute focus to be placed on understanding the inner workings of the GA architecture in a hands-on approach prior to investigating a specific implementation.

After verifying a GA architecture that has potential to achieve desirable results, the simple model is replaced with a more computationally intensive and physiologically relevant CD model. Finally, an exhaustive search algorithm is used with the same CD Model and optimization function to perform two tasks: First, to gauge GA performance, and second, as a parallel investigation to answer the overall question of whether or not adjustments to regional firing times can reduce the impacts of mechanical cardiac dyssynchrony, on measured acceleration CD.

In the case of the 2 parameter searches, the GA results came close but not quite as good as those of the exhaustive searches when averaging multiple separate GA runs (Tables 3 and 5 along with Appendices A and B). The results are as expected since GAs, by nature, get close to the optimal solution for any one particular run, but rarely achieve the global optimum without trial averaging or other method of fine tuning to create a more hybrid search architecture.

In the case of the 4 parameter searches, the GA results are infeasible when compared to their respective exhaustive search results (Tables 4 and 6 along with Appendices A and B). Even

firing all the regions simultaneously is better in many cases as can be seen when comparing identical trial conditions as noted in Tables 3, 4, 7, and 8 with Figures in Appendices A and C. A simple way to remedy this might be to selectively place this sequence into the initial GA population; however, other modification will also likely be required to avoid early convergence in the case that this solution is significantly more fit than other initial solutions.

GAs can likely achieve the results desired in using cardiac wall acceleration to reduce CD, but the question is at what cost computationally. In the situation where the GA is properly tuned, there is a chance that computational performance could be drastically better than a similarly performing exhaustive search. For example, in a 2 parameter, no dyssynchrony test run, a single instance of the GA search executed in about $1/8^{th}$ the time of a single run of the exhaustive search. While these single execution times vary for the 4 parameter search, there is still substantial margin between GA and exhaustive searches that can be exploited to obtain a similar solution from the GA, in less time than a sufficiently narrow exhaustive search if the GA can be tuned to reliably find the desired result in a single run.

Due to the discussed flexibility of GA implementation, the GA shows strong promise in areas of medical/biological interaction and optimization given proper tuning. In contrast however, significant tuning must yet be done in the case of using GAs to assign firing times for a pacemaker to provide a higher chance of convergence to a near optimal solution. In addition, use of the GA algorithm against more comprehensive cardiac models is highly recommended since it is suspected that discussed model shortfalls may substantially change the search space of the optimization function.

Already cautioned in [32], it is advised against GAs operating directly on a system in on-line applications due to the extreme randomness in how GA solutions are generated. The

exception to this would be in the case where a system is known to be robust enough to tolerate what could be significant stress on the system by very poor solution choices. Since the system under test in this case would be a human heart, it is inadvisable to allow a GA unhindered search ability across a global solution space. Strong safeguards such as narrowing the search range for any given generation should be considered to protect human life.

Problems strongly suited for GA optimization are usually poorly understood, or have poorly behaved search spaces. This paradox largely confines GA use to off-line applications and on-line applications where a short hiccup will not cause catastrophic damage. In the case of medical applications, it is questionable to allow such unrestricted experimentation of the device directly on a patient's heart. Likely, the solution to this is twofold. First, an individualized model could be created and run as a secondary operation within a pacemaker for each patient. This would allow on-line timing changes only after obtaining one or multiple "best fits" from model driven GA executions. Second, a hybrid system, similar to those discussed in [32], utilizing all or portions of the GA architecture modified and combined with another type of optimization, such as exhaustive search, could be implemented to help quell the randomness of the initial GA search space and allow a slower, more controlled path toward convergence and the globally optimal solution.

For systems with well behaved and optimization functions and well understood search spaces, utilizing the inherent parallel operations of the GA to find suitable starting positions and systematically "climbing the hill" to the local optimum from a given starting point could speed the process of finding the local optimum from each starting point, yet still yield a global optimum when taking those now optimized starting points to perform the rest of the GA protocol.

7.2. Exhaustive Search Conclusions

While it is slow, especially with higher degrees of freedom, the exhaustive search has easily shown promise in using cardiac wall acceleration as a metric of measuring and correcting CD. In all trials of importance, exhaustive search out performed both GA search, and "no adjustment" testing conditions. Timing adjustment has been shown to improve acceleration CD by up to 56%, increase both the dyssynchronous region's workload and instantaneous power generation by up to 50%, and decrease the healthy regions' workload by up to 15% and instantaneous power generation by up to 7%. Exhaustive searching gives merit to the continued investigation of using cardiac wall acceleration as a metric for both measuring, and reducing CD on an individualized basis.

Even with exhaustive search being able to find the best solution, it also finds the worst solution in an unrestricted search space. Prior to non-model trials, restrictions will need to be placed on the search space available to exhaustive or any other type of search.

7.3. Future Work

This thesis provides an in depth tutorial on GA operation and problem casting along with a first step examination of the merits and demerits of utilizing GA and exhaustive search in a medical device control system applications. The focus was placed on proposing an improved method of more systematically adjusting a cardiac pacemaker lead timings to reduce measured ventricular acceleration CD caused by both mechanical and electrical cardiac in a model based, MATLAB environment. While these investigations include the use of GAs and exhaustive search methods, safety in the random navigation of the GA search space and its results on the heart have not been investigated and were not considered a concern for this specific implementation.

Medical devices are understandably a highly regulated industry and while that regulation ultimately leads to safer devices, at this stage of development, designing to those stringent criteria would unduly slow the gathering of beneficial results at early stages of development. Therefore, future investigations should strongly consider the safety precautions necessary for clinical trials as research pushes closer to actual device implementation.

In addition, to the short comings discussed in sections 7.2 and 7.2.1, the model only provides a 1-dimensional acceleration for each of the regions of interest where a real heart would have 3-dimensional accelerations. As an example, future work would have to allow for 3-dimensional accelerations and other complexities inherent to truer cardiac models as a path toward eventual hardware implementation, implantation, and testing. Future research could attempt to streamline either or both of these fronts to improve eventual patient safety in the search for a system that already shows strong promise in realizing an improved solution over firing all regions simultaneously.

One option for future GA exploration may be in constraining the explorable search space of the GA to a smaller region around the initial conditions given by either the model or physiological measurement. There are multiple reasons for this, but the most important being that if this GA is allowed to make on-line changes to an individual's cardiac timing it must be in a more controlled fashion than what is inherent with traditional GA operation. Traditional GAs have the potential to explore the entire search space given to them; however, in the case of cardiac pacing, it is unknown if one wrong pacing trial can send a heart into arrest, but it is something that cannot be risked. Thus, creating a hybrid GA with other optimization properties may be a better solution. One example may be starting with an exhaustive search and using those results to then constraining exploration of a GA to an area close to the initial

measurements, while slowly tracking that search window with the current best solution could help mitigate the possibility of having a catastrophic timing combination causing cardiac arrest. Another reason this could be beneficial, is it allows an easy path toward allowing the GA to track with cardiac changes over time in that, this search and slow movement concept can be looped indefinitely and allow for change in timing that would be less noticeable to the patient unlike traditional GA or exhaustive searches that test the entire scope of the solution space when restarted.

Another area of possible improvement lies in how GA solutions are utilized. Following recommendations in [14], multiple GA runs were performed and averaged for a given set of trial conditions. These referenced cases involve non-periodic signals, but the tests performed here are on periodic waveforms. In this case, strict averaging caused trials to appear worse than they may actually be if convergence is not guaranteed to be in a relatively small window. A better approach may be to modify the averaging to take advantage of the periodic nature of the waveforms, perform cluster recognition and ignore outliers prior to averaging, or to simply pick the best, final solution of the individual GA runs. The latter approach is just picking the best solution from parallel, isolated GA experiments. Instead, it may be better to focus on selecting an approach that does not require as much computing power. Rather, it may be more beneficial to focus on making smarter choices in GA parameter selection and windowing to allow for more rapid and clustered convergence. Likely, experimentation will be required to find the correct balance of GA starting parameters, search windows, and forced starting conditions to generate better solutions.

This work takes the first steps toward creating a system that can continually adjust as physiological changes take place in the heart to ensure an up-to-date, near optimal solution for

every patient implanted with what could very likely become the next generation in both cardiac resynchronization therapy, and individualized medicine.

8. WORKS CITED

[1] F. W. Prinzen, *et. al.*, "Redistribution of Myocardial Fiber Strain and Blood Flow by Asynchronous Activation," Am. J. Physiol. Heart Circ. Physiol., vol. 259, no. 2, pp. H300-308, Aug., 1990.

[2] M. F. M. Oosterhout, *et. al.*, "Asynchronous Electrical Activation Induces Asymmetrical Hypertrophy of the Left Ventricular Wall," Circulation., vol. 96, no. 6, pp. 588-95, 1998.

[3] D. D. Spragg, *et. al.*, "Regional alterations in protein expression in the dyssynchronous failing heart," Circulation., vol. 108, no. 8, pp. 929-932, 2003.

[4] J. B. Thambo, *et. al.*, "Detrimental ventricular remodeling in patients with congenital complete heart block and chronic right ventricular apical pacing," Circulation., vol. 110, no. 25, pp. 3766-3772, 2004.

[5] O. Aquilina, A Brief History of Cardiac Pacing," Images Paediatr Cardiol., vol. 8, no. 2, pp. 17-81, 2006.

[6] L. F. Tops, M. J. Schalij, and J. J. Bax, "The Effects of Right Ventricular Apical Pacing on Ventricular Function and Dyssynchrony," J. Am. Coll. Cardiol., vol. 59, no. 9, pp. 764-776, 2009.

[7] D. E. Golberg, "Genetic algorithms in search, optimization, and machine learning," Addion Wesley, 1989.

[8] J. Carr, "An Introduction to Genetic Algorithms," [Online]. pp. 1-40, Available: https://karczmarczuk.users.greyc.fr/TEACH/IAD/GenDoc/carrGenet.pdf

[9] M. Obitko, P. Slavik, and W. Patzold, "GENETIC ALGORITHMS, Introduction to Genetic Algorithms," [Online]. Available: http://www.obitko.com/tutorials/genetic-algorithms/index.php

[10] D. Whitley, "An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls," Inform. Software. Tech., vol. 43, no. 14, pp. 817-831, 2001.

[11] R. L. Haupt, "Optimum Population Size and Mutation Rate for a Simple Real Genetic Algorithm That Optimizes Array Factors," in IEEE Antennas Propagat. Soc. Int. Symp. Transmitting Waves of Progress to the Next Millenium., 2000.

[12] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, Univ. of Michigan, Ann Arbor, MI, 1975.

[13] G. Liu, and J. Chen, "The Application of Genetic Algorithm Based on Matlab in Function Optimization," 2011 Int. Conf. on Electrical and Control Engineering, 2011.

[14] R. Wehrens, and L. M. C. Buydens, "Evolutionary optimisation: a tutorial," Trends Anal. Chem., vol. 17, no. 4, pp. 193-203, 1998.

[15] Y. J. Cao, and Q. H. Wu, "Teaching genetic algorithm using MATLAB," Int. J. Elec. Eng. Educ., vol. 36, no. 2, pp. 139-153, 1999.

[16] L. F. Tops, *et. al.*, "Right Ventricular Pacing Can Induce Ventricular Dyssynchrony in Patients With Atrial Fibrillation After Atrioventricular Node Ablation," J. Am. Coll. Cardiol., vol. 48, no. 8, pp. 1642-1648, 2006.

[17] A. S. Manolis, "The Deleterious Consequences of Right Ventricular Apical Pacing: Time to Seek Alternate Site Pacing," Pace., vol. 29, no. 3, pp. 298-315, 2006.

[18] C. R. Reeves, "Genetic Algorithms and Statistical Methods: A Comparison," 1st Int. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995.

[19] S. Gotshall, and B. Rylander, "Optimal Population Size and the Genetic Algorithm." [Online.] Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.417.1552&rep=rep1&type=pdf

[20] M. Dianati, I. Song, and M. Treiber, "An Introduction to Genetic Algorithms and Evolution Strategies," [Online.] Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.6910&rep=rep1&type=pdf

[21] L. Sewanan, Modeling Cardiac Muscle Mechanics," Undergraduate thesis, Trinity College, Hartford, CT, 2012. Trinity College Digital Repository, <http://digitalrepository.trincoll.edu/theses/213>.

[22] D. Whitley, "A genetic algorithm tutorial," Stat. Comput., vol. 4, no. 2, pp. 65-85, 1994.

[23] P. A. Diaz-Gomez, and D. F. Hougen, "Initial Population for Genetic Algorithms: A Metric Approach," GEM. 2007. Available: http://www.cameron.edu/~pdiaz-go/GAsPopMetric.pdf

[24] O. Roeva, S. Fidanova, and M. Paprzycki, "Influence of the population size on the genetic algorithm performance in case of cultivation process modeling," Computer Science and Information Systems (FedCSIS), 2013 Federated Conf. on IEEE, 2013.

[25] L. Ganjehei, M. Razavi, and A. Massumi, "Cardiac Resynchronization Therapy: A Decade of Experience and the Dilemma of Nonresponders," Tex. Heart. I. J., vol. 38, no. 4, pp. 358-360, 2011.

[26] J. G. F. Cleland, *et. al.*, "The effect of cardiac resynchronization on morbidity and mortality in heart failure," New. Engl. J. Med., vol. 352, no. 15, pp. 1539-1549, 2005.

[27] F. W. Prinzen, *et. al.*, "Asymmetric Thickness of the Left Ventricular Wall Resulting from Asynchronous Electric Activation: A Study in Dogs with Ventricular Pacing and in Patients with Left Bundle Branch Block," Am. Heart. J., vol. 130, no. 5, pp. 1045-1053.

[28] S. Sagar, *et. al.*, "Effect of Long-Term Right Ventricular Pacing in Young Adults With Structurally Normal Heart," Circulation., vol. 121, no. 15, pp. 1698-1705, 2010.

[29] K. Vernooy, *et. al.*, "Ventricular remodeling during long-term right ventricular pacing following His bundle ablation," Am. J. Cardiol., vol. 97, no. 8, pp. 1223-1227, 2006.

[30] H. E. D. J. Ter Keurs, W. H. Rijnsburger, R. Van Heuningen, and M. J. Nagelsmit, "Tension Development and Sarcomere Length in Rat Cardiac Trabeculae: Evidence of Length-Dependent Activation," Cardiac Dynamics Developments, Cardiovasc. Med., pp. 25-36, 1980.

[31] J. Sadoshima, *et. al.*, "Molecular characterization of the stretch-induced adaptation of cultured cardiac cells. An in vitro model of load-induced cardiac hypertrophy," J. Biol. Chem., vol. 267, no. 15, pp. 10551-10560, 1992.

[32] P. J. Fleming, and C. M. Fonseca, "Genetic algorithms in control systems engineering: A brief introduction," Genetic Algorithms for Control Systems Engineering, IEE Colloq. on. IET, 1993.

[33] K. Toischer, *et. al.*, "Differential cardiac remodeling in preload versus afterload," Circulation, vol. 122, no. 10, pp. 993-1003, 2010.

[34] Pacemaker NIH.jpg. Digital image. Wikimedia Commons. National Heart Lung and Blood Institute (NIH), 12 Nov. 2013. Web. <https://commons.wikimedia.org/wiki/File:Pacemaker_NIH.jpg>.

[35] Deglr6328~commonswiki. Skeletal Muscle.jpg. Digital image. Wikimedia Commons. N.p., 11 Sept. 2005. Web. <https://commons.wikimedia.org/wiki/File:Skeletal_muscle.jpg>.

APPENDIX A. PLOTS OF GA SEARCH VARIATIONS FOR CD MODEL

Note: Labeling convention for acceleration plots (x, y) where x is the number of fixed regions, and y is the number of adjusted regions.

A.1. No Dyssynchrony

Case 1: Scripted_Initialization_no_dyss.m parameters

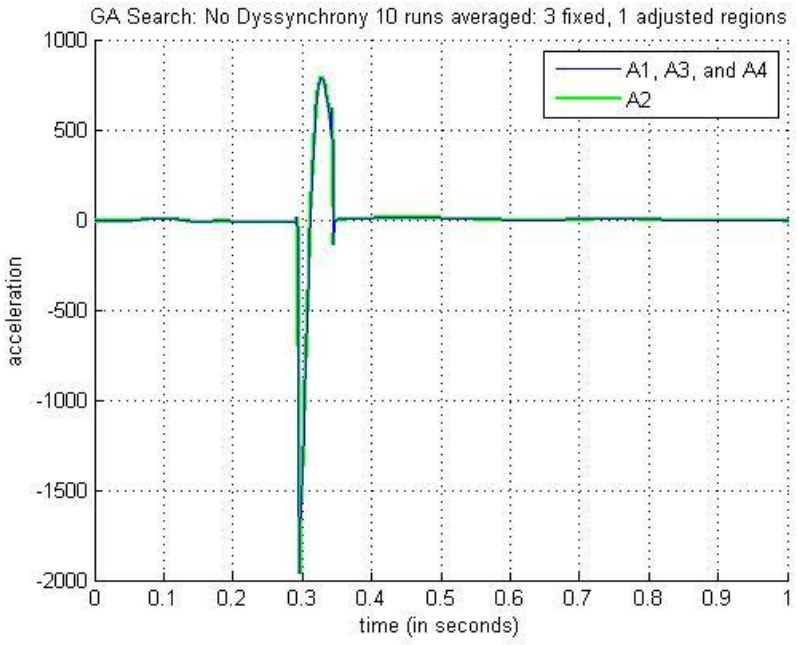

Figure A-1: No Dyssynchrony (3, 1) Acceleration for GA Search 10 runs averaged

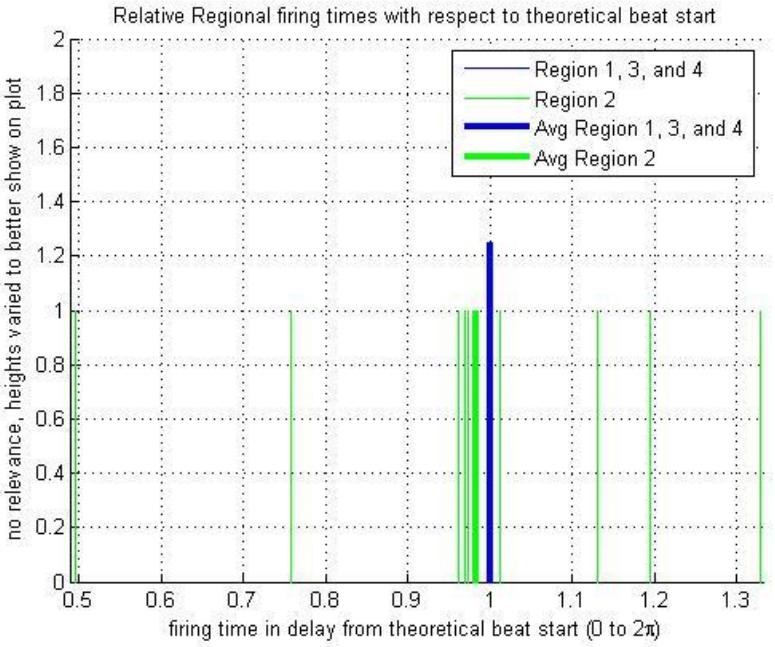

Figure A-2: No Dyssynchrony (3,1) Timing for GA Search 10 runs

Case 2: Scripted_Initialization_no_dyss.m parameters

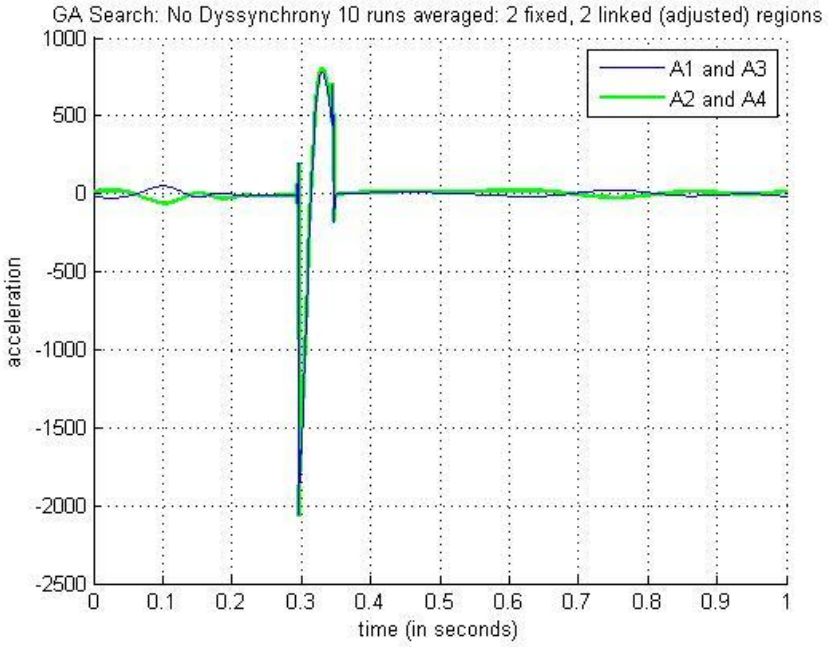

Figure A-3: No Dyssynchrony (2, 2) Acceleration for GA Search 10 runs averaged

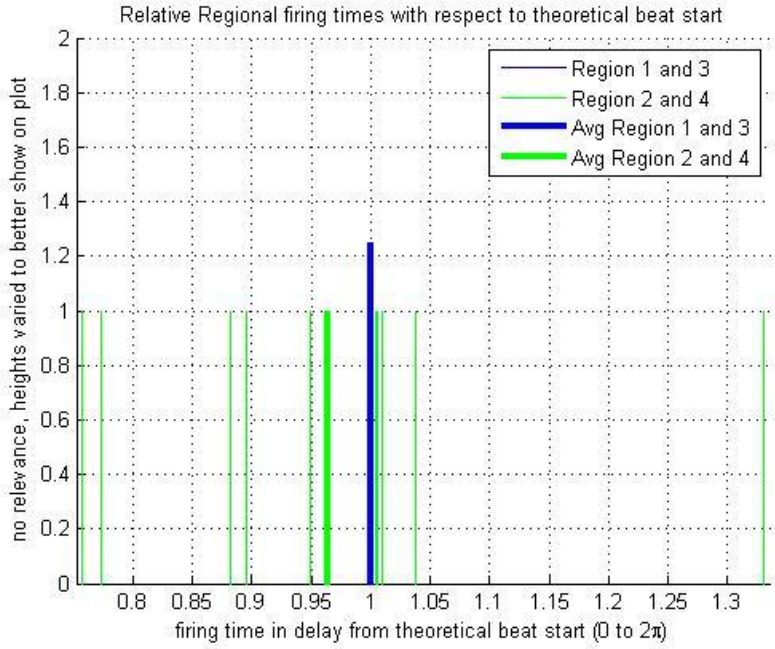

Figure A-4: No Dyssynchrony (2,2) Timing for GA Search 10 runs

Case 3: Scripted_Initialization_no_dyss.m parameters

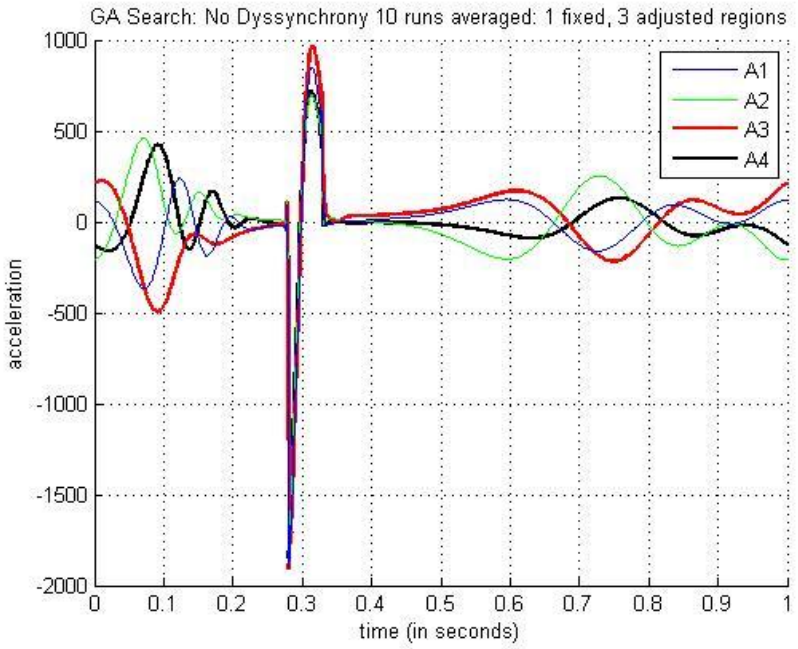

Figure A-5: No Dyssynchrony (1, 3) Acceleration for GA Search 10 runs averaged

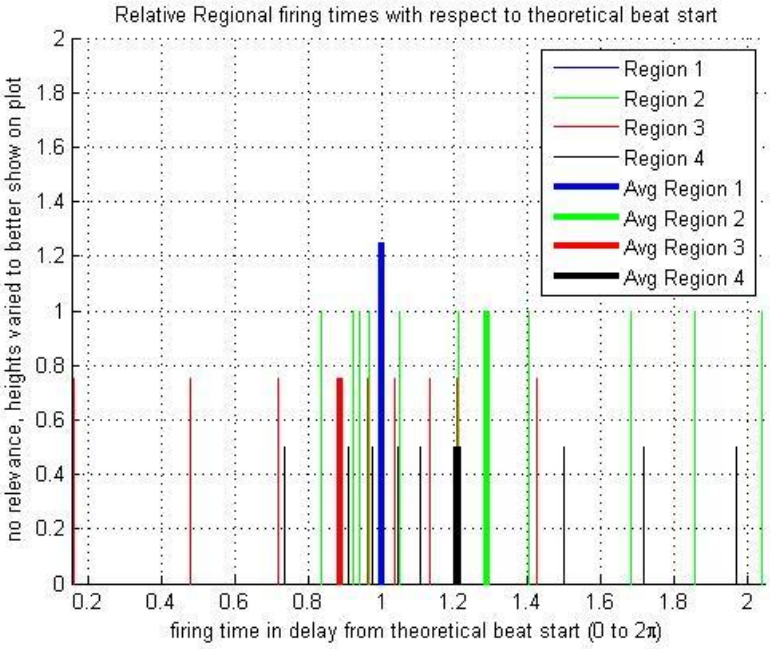

Figure A-6: No Dyssynchrony (1,3) Timing for GA Search 10 runs

A.2. Resistance Dyssynchrony

Case 1: Scripted_Initialization_r2_dyss_0_015.m parameters

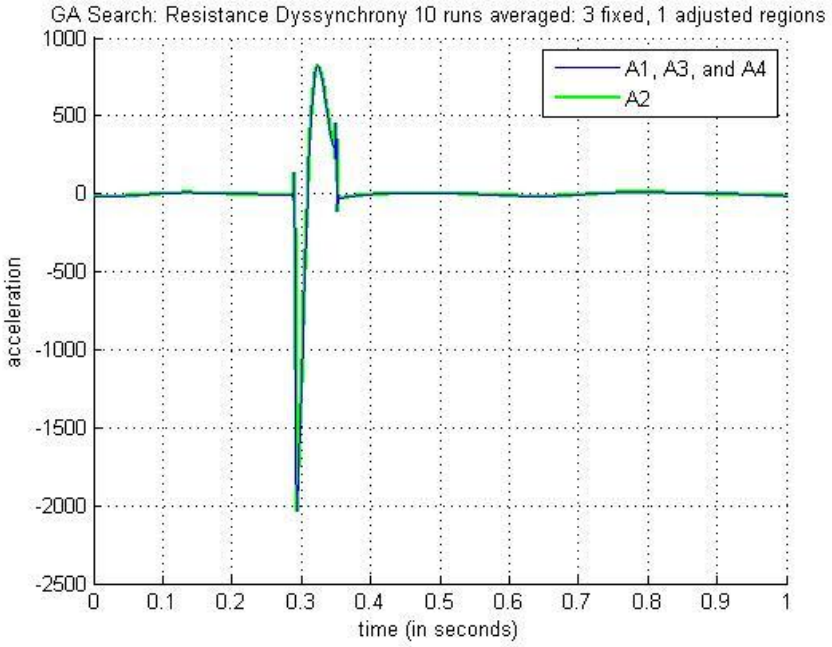

Figure A-7: Resistance Dyssynchrony (3, 1) Acceleration for GA Search 10 runs averaged

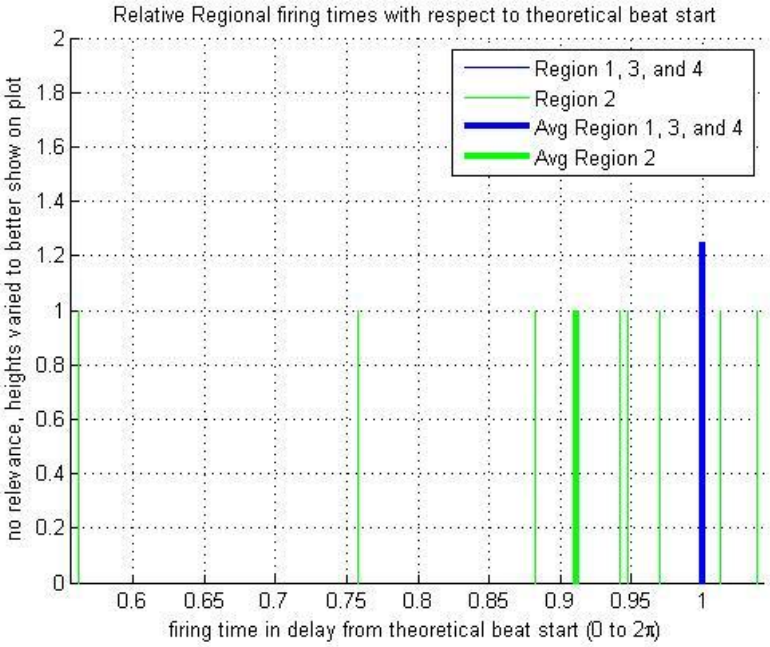

Figure A-8: Resistance Dyssynchrony (3,1) Timing for GA Search 10 runs

Case 2: Scripted_Initialization_r2_dyss_0_015.m parameters

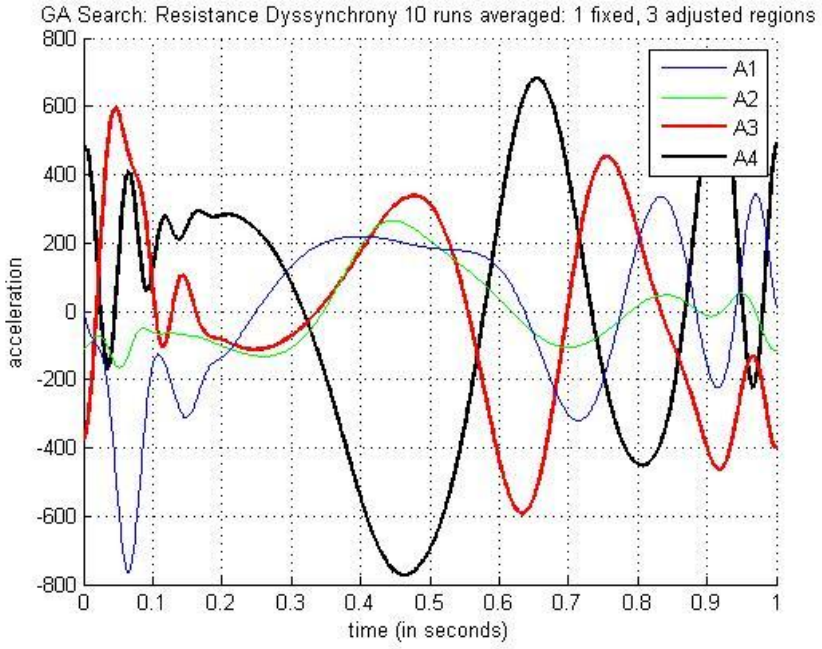

Figure A-9: Resistance Dyssynchrony (1, 3) Acceleration for GA Search 10 runs averaged

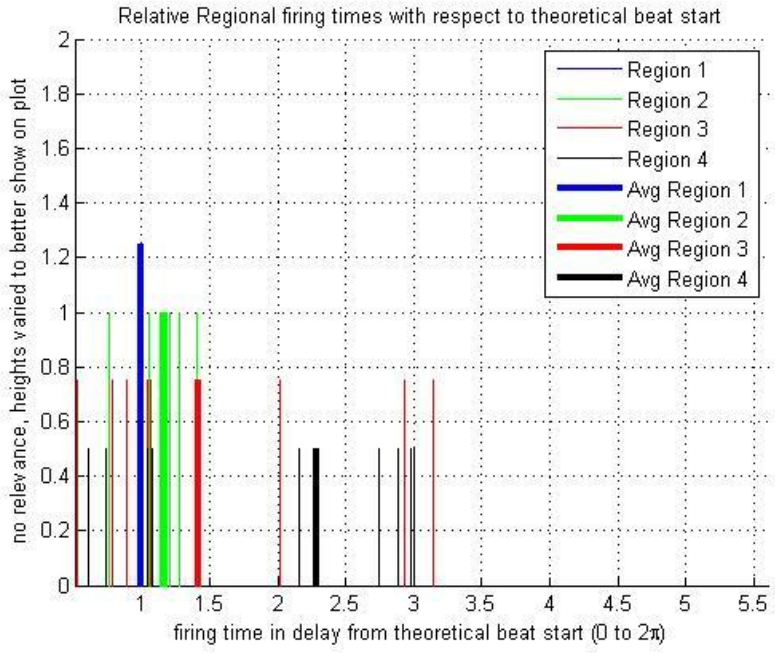

Figure A-10: Resistance Dyssynchrony (1,3) Timing for GA Search 10 runs

A.3. Mass Dyssynchrony

Case 1: Scripted_Initialization_m2_dyss_0_01.m parameters

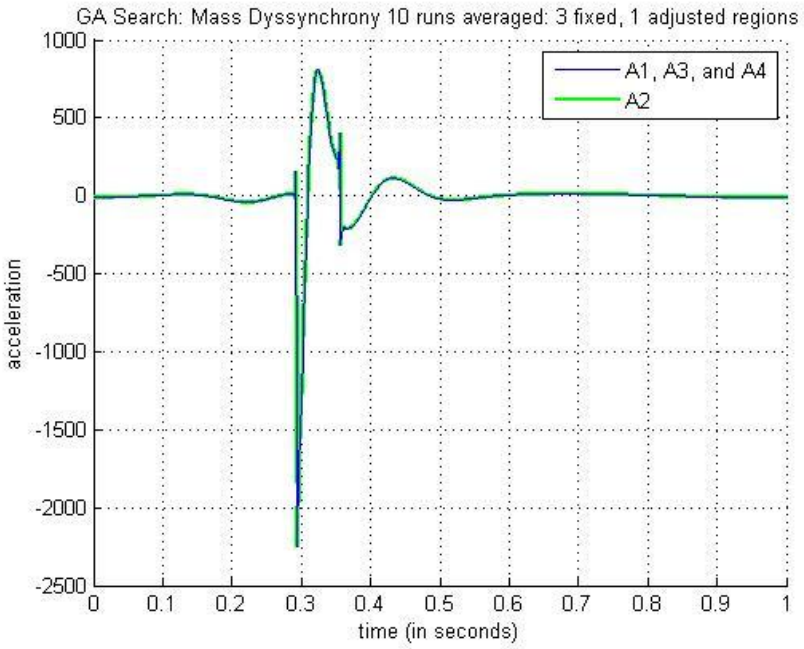

Figure A-11: Mass Dyssynchrony (3, 1) Acceleration for GA Search 10 runs averaged

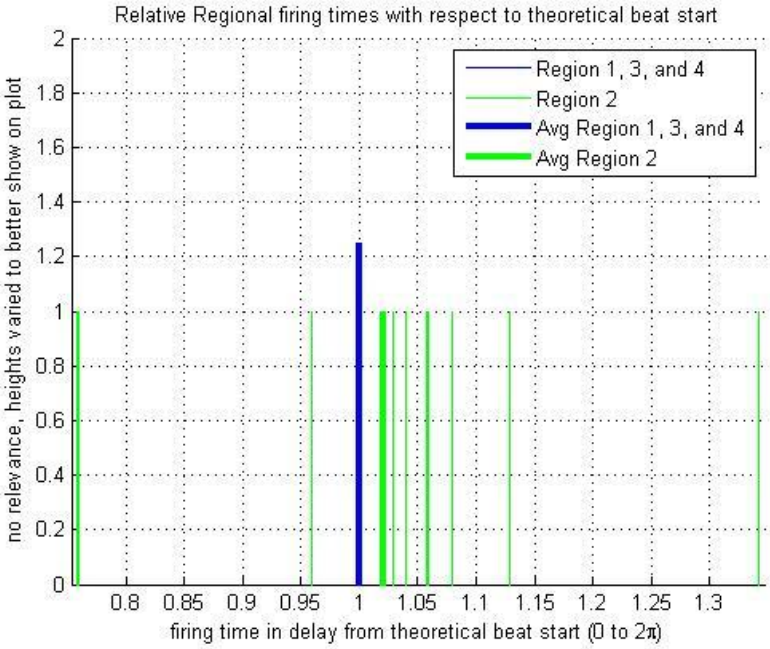

Figure A-12: Mass Dyssynchrony (3,1) Timing for GA Search 10 runs

Case 2: Scripted_Initialization_m2_dyss_0_01.m parameters

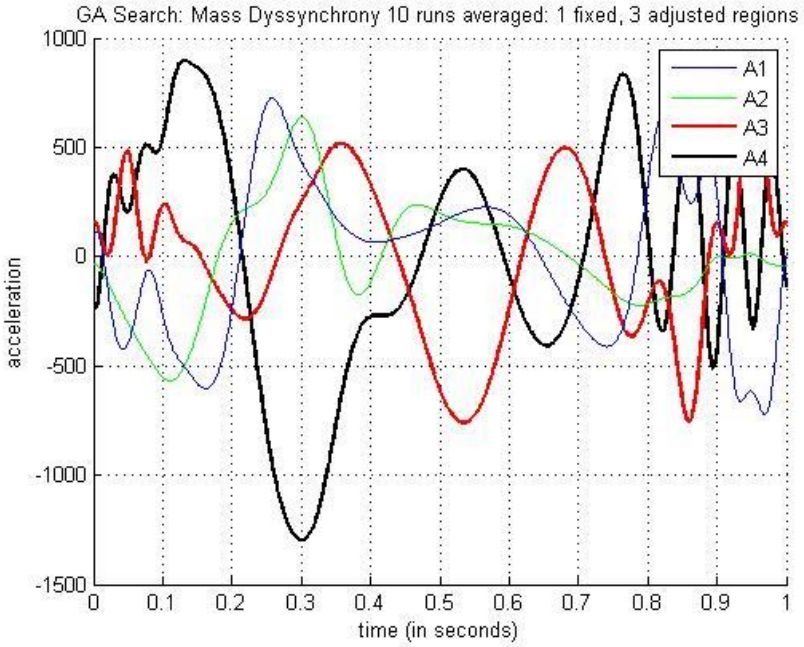

Figure A-13: Mass Dyssynchrony (1, 3) Acceleration for GA Search 10 runs averaged

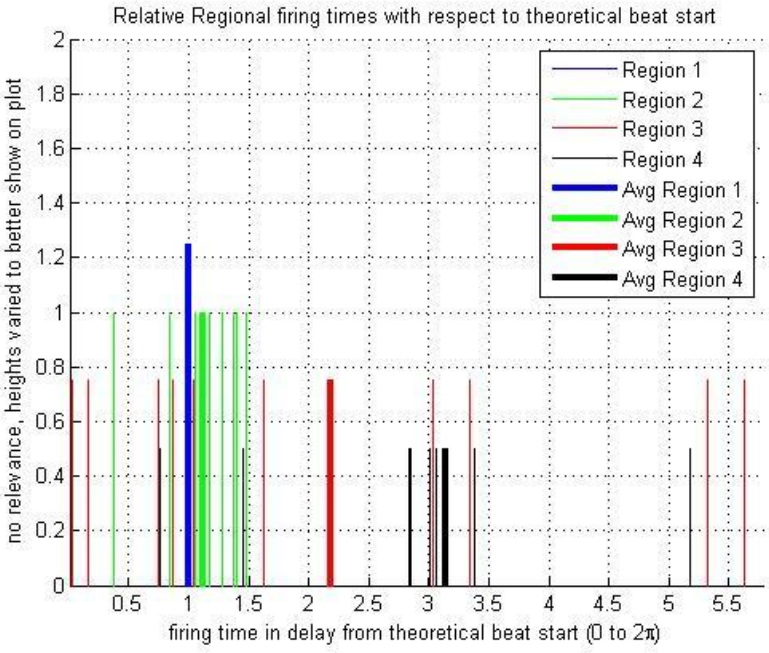

Figure A-14: Mass Dyssynchrony (1,3) Timing for GA Search 10 runs

A.4. Minimum Elastance Dyssynchrony

Case 1: Scripted_Initialization_min_elas2_dyss_4.m parameters

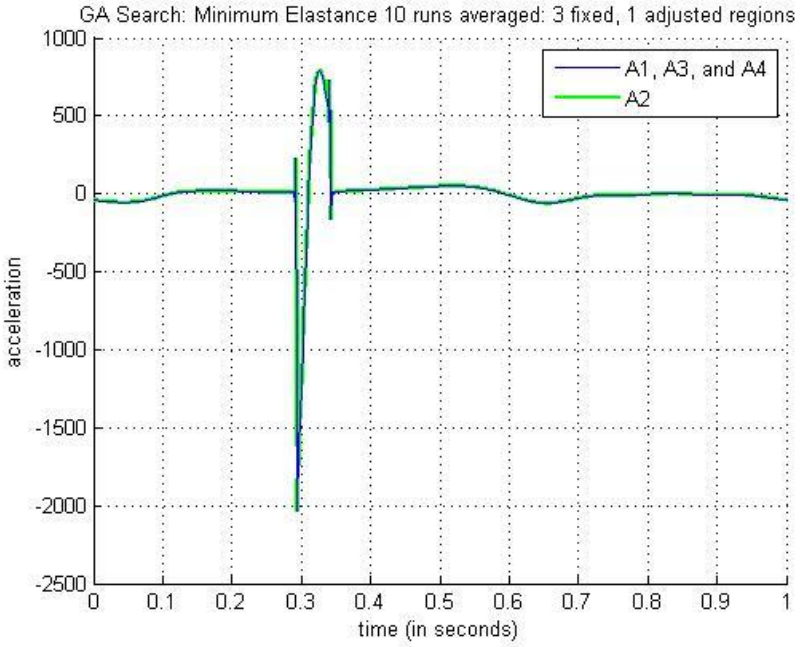

Figure A-15: Min Elastance Dyssynchrony (3, 1) Acceleration for GA Search 10 runs avg

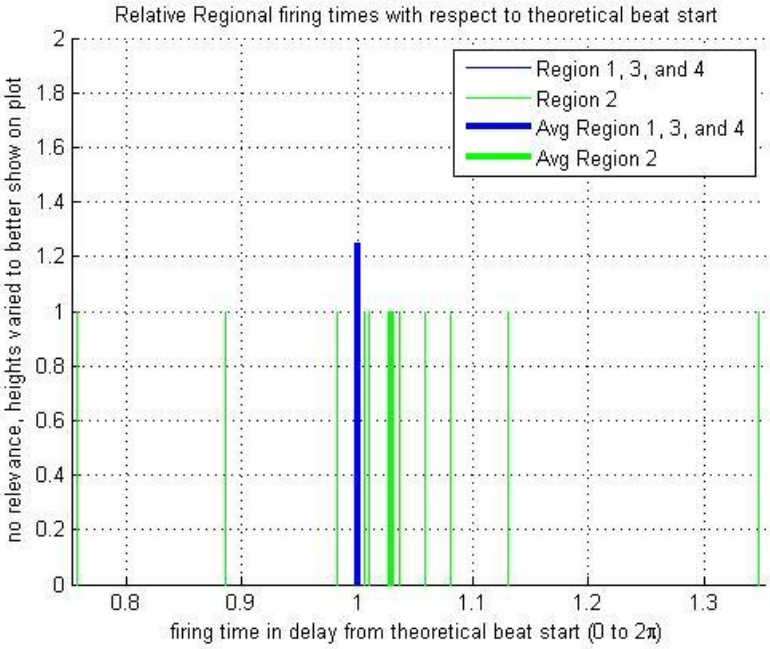

Figure A-16: Min Elastance Dyssynchrony (3,1) Timing for GA Search 10 runs

Case 2: Scripted_Initialization_min_elas2_dyss_4.m parameters

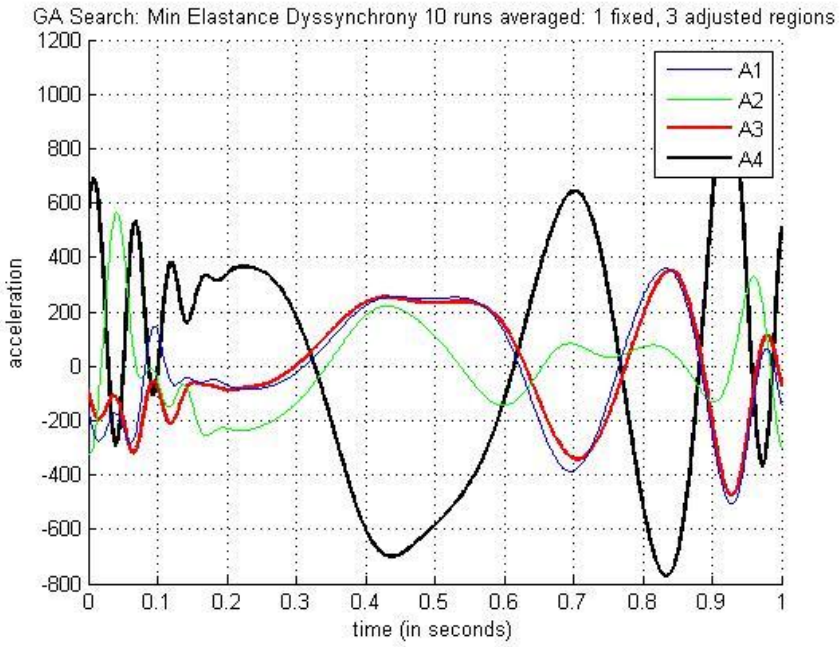

Figure A-17: Min Elastance Dyssynchrony (1, 3) Acceleration for GA Search 10 runs avg

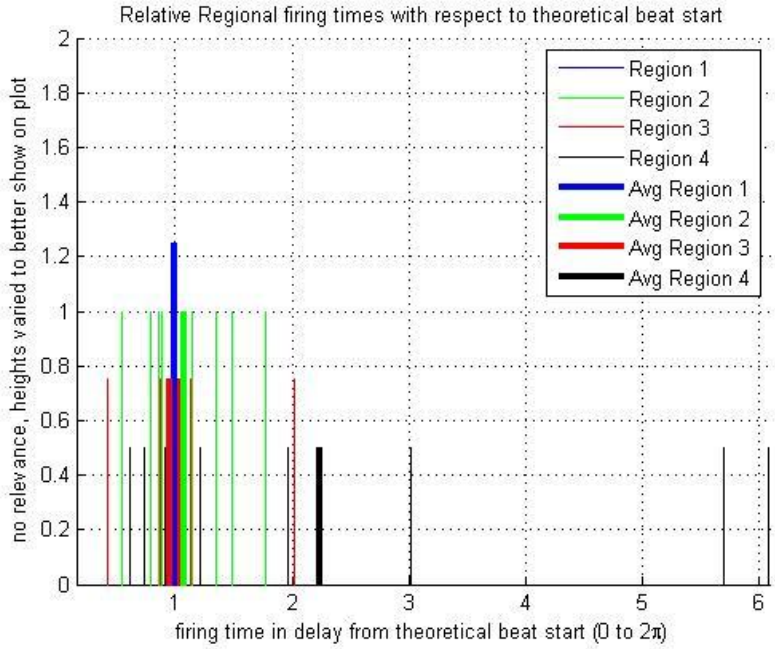

Figure A-18: Min Elastance Dyssynchrony (1,3) Timing for GA Search 10 runs

A.5. Maximum Elastance Dyssynchrony

Case 1: Scripted_Initialization_max_elas2_dyss_40.m parameters

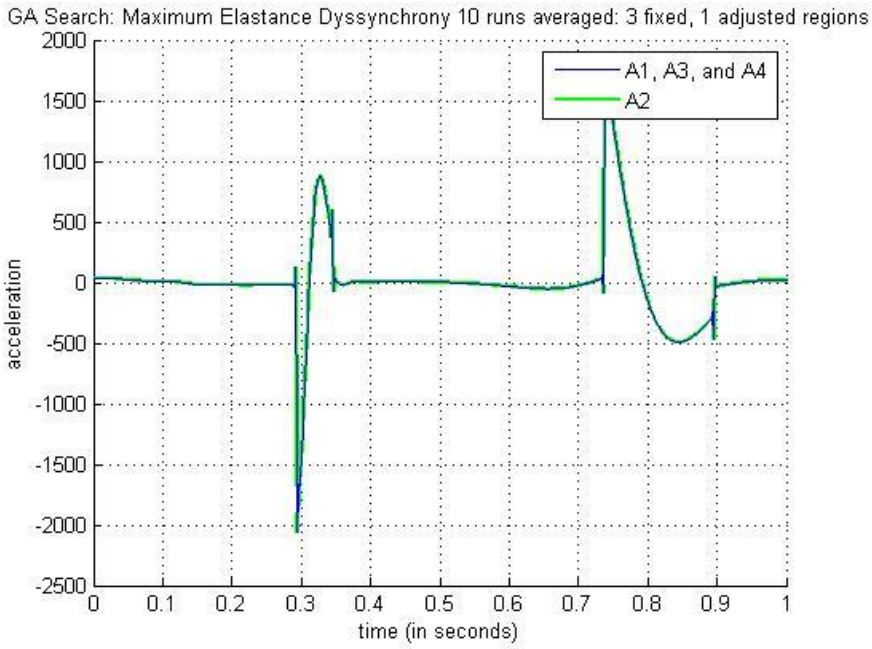

Figure A-19: Max Elastance Dyssynchrony (3, 1) Acceleration for GA Search 10 runs avg

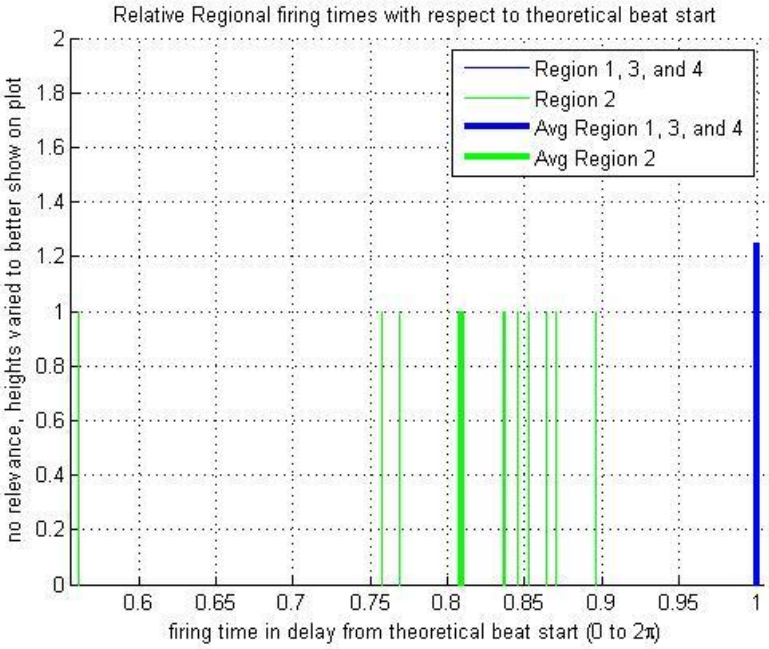

Figure A-20: Max Elastance Dyssynchrony (3,1) Timing for GA Search 10 runs

Case 2: Scripted_Initialization_max_elas2_dyss_40.m parameters

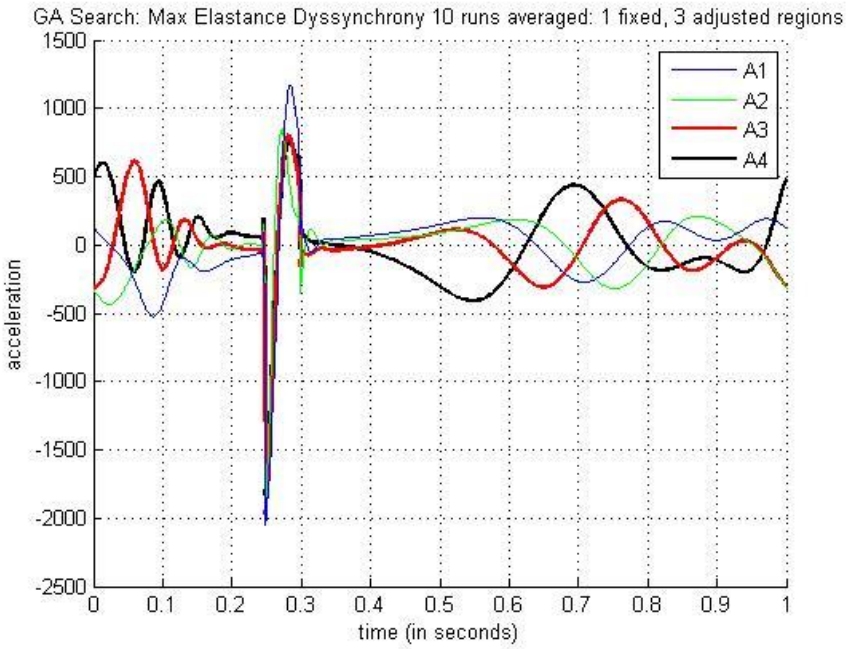

Figure A-21: Max Elastance Dyssynchrony (1, 3) Acceleration for GA Search 10 runs avg

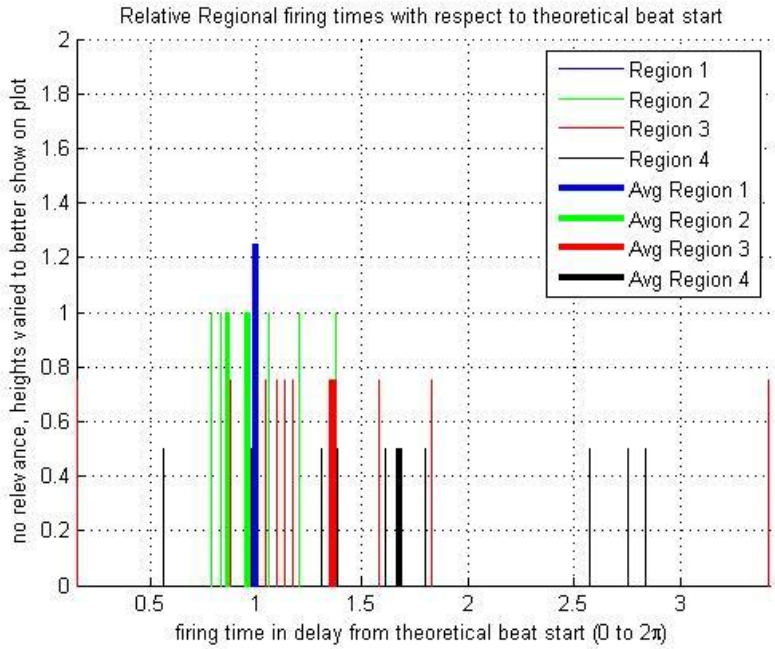

Figure A-22: Minimum Elastance Dyssynchrony (1,3) Timing for GA Search 10 runs

A.6. Population Size Variation: No Dyssynchrony

Scripted_Initialization_no_dyss.m parameters

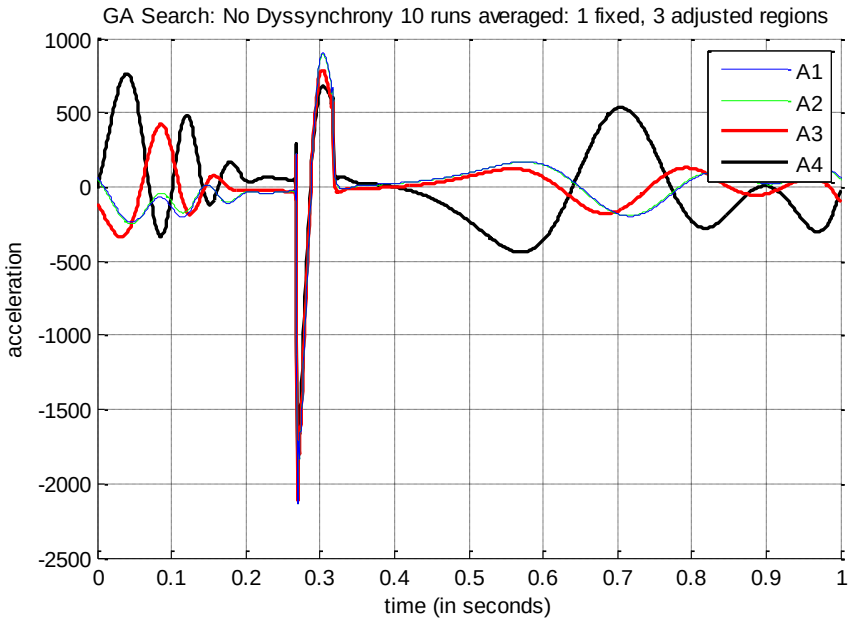

Figure A-23: No Dyssynchrony (1, 3) Acceleration for GA Search 10 Runs Avg, Pop Size = 48

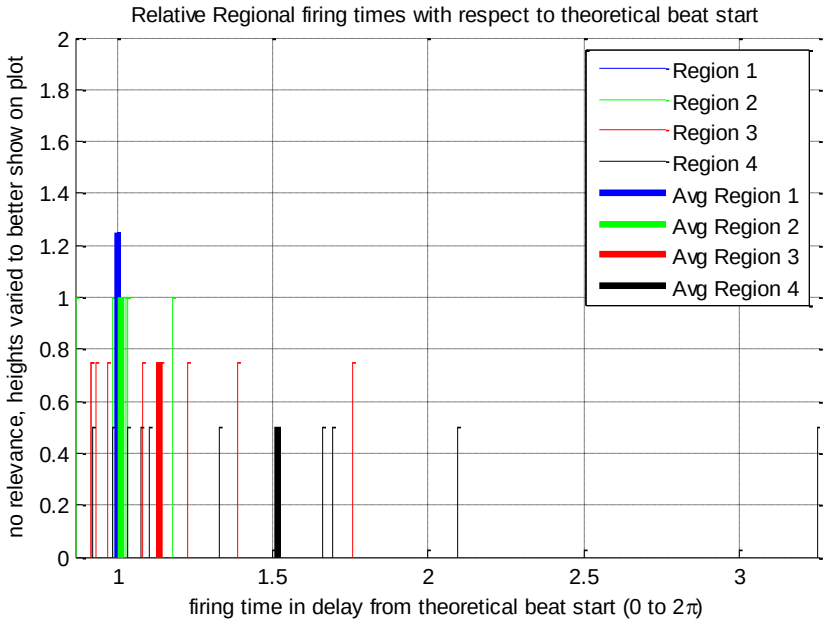

Figure A-24: No Dyssynchrony (1,3) Timing for GA Search 10 Runs, Population Size = 48

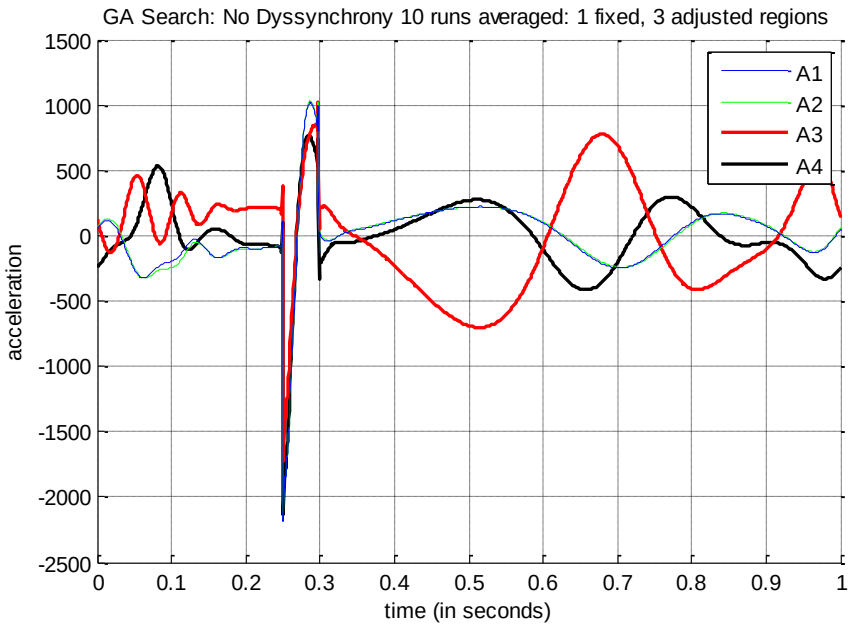

Figure A-25: No Dyssynchrony (1, 3) Acceleration for GA Search 10 Runs Avg, Pop Size = 72

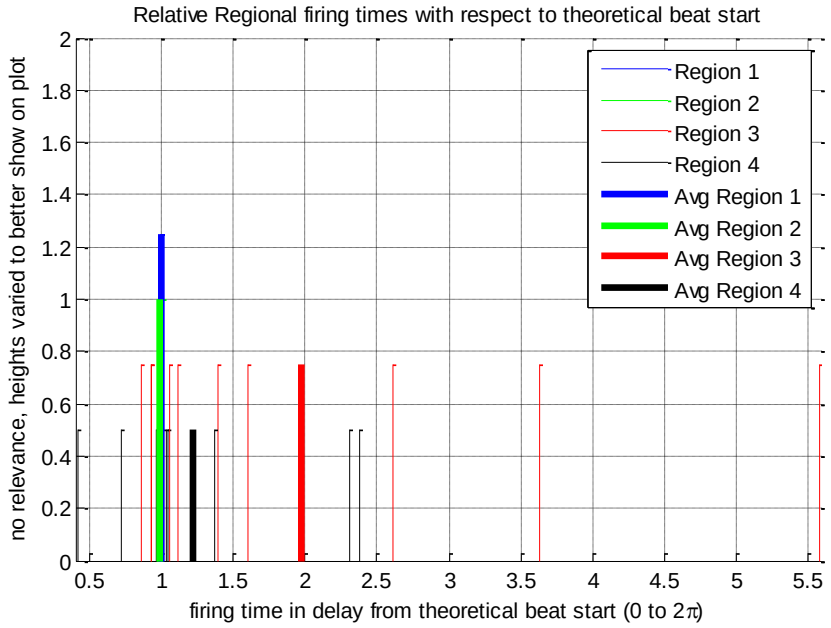

Figure A-26: No Dyssynchrony (1,3) Timing for GA Search 10 Runs, Pop Size = 72

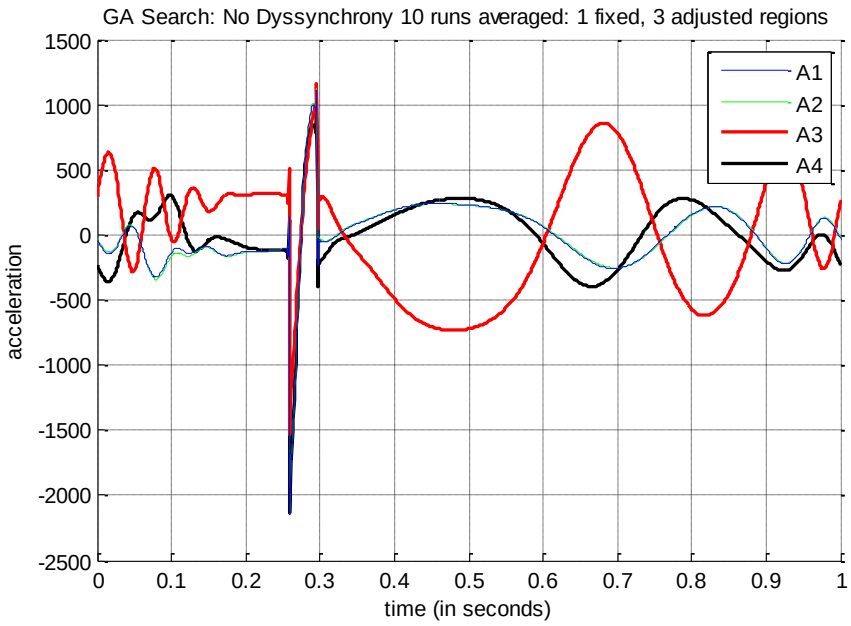

Figure A-27: No Dyssynchrony (1, 3) Acceleration for GA Search 10 Runs Avg, Pop Size = 96

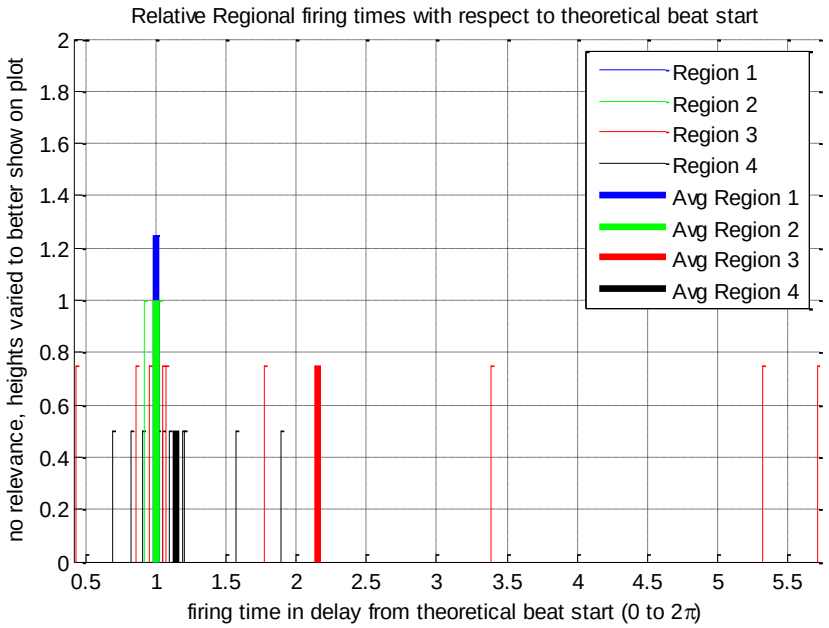

Figure A-28: No Dyssynchrony (1,3) Timing for GA Search 10 Runs, Pop Size = 96

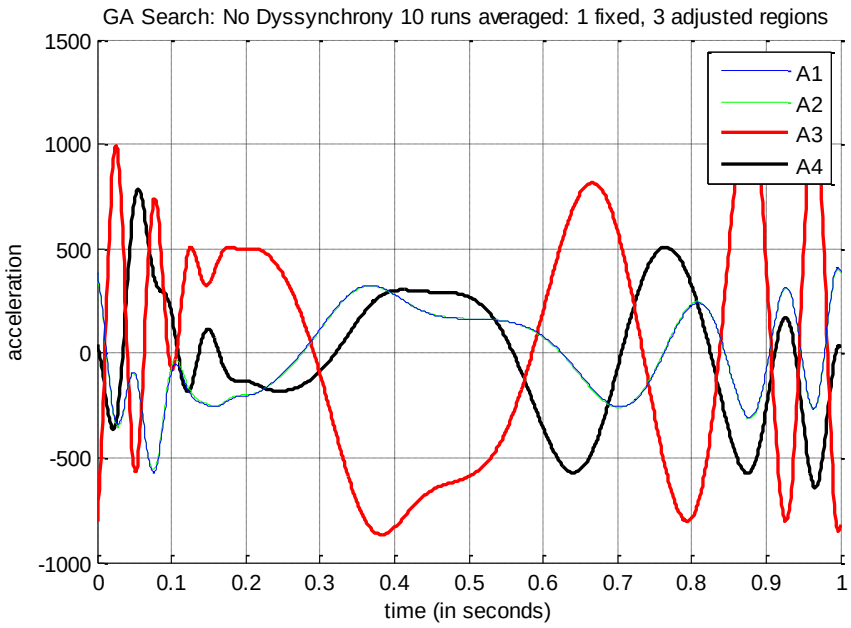

Figure A-29: No Dyssynchrony (1, 3) Acceleration for GA Search 10 Runs Avg, Pop Size = 120

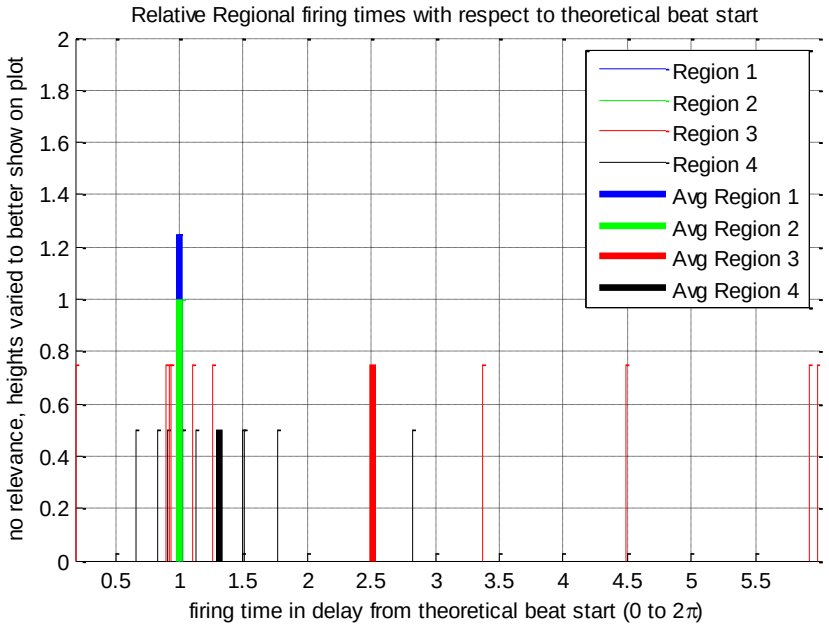

Figure A-30: No Dyssynchrony (1,3) Timing for GA Search 10 Runs, Pop Size = 120

A.7. Mutation Rate Variation: No Dyssynchrony

Scripted_Initialization_no_dyss.m parameters

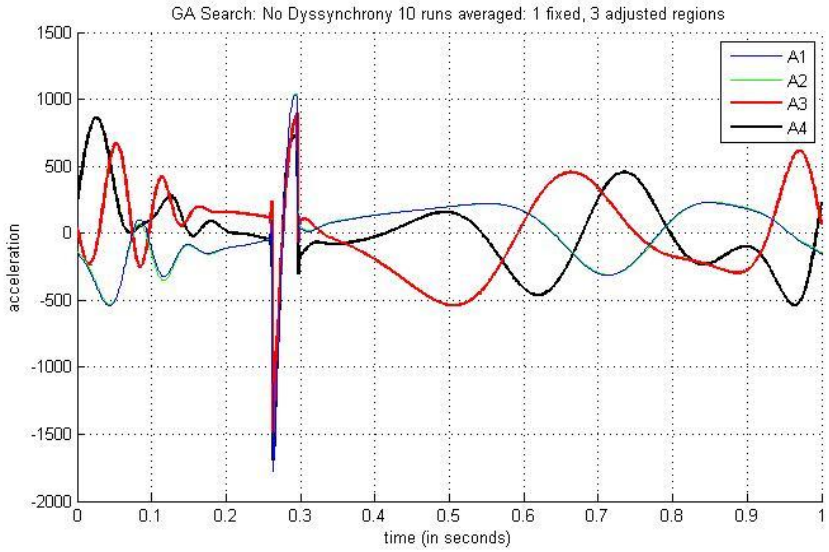

Figure A-31: No Dyssynchrony (1, 3) Acceleration for GA Search 10 Runs Avg, Mutation Rate = 0.01 (1%)

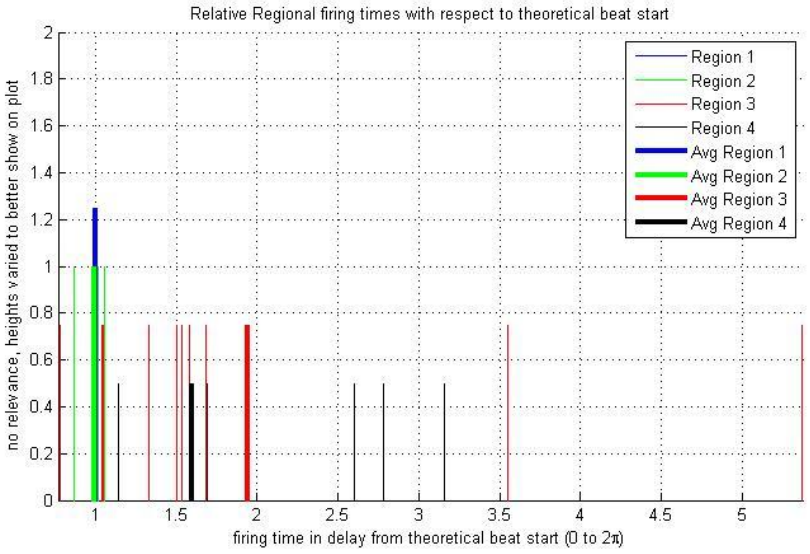

Figure A-32: No Dyssynchrony (1,3) Timing for GA Search 10 Runs, Mutation Rate = 0.01 (1%)

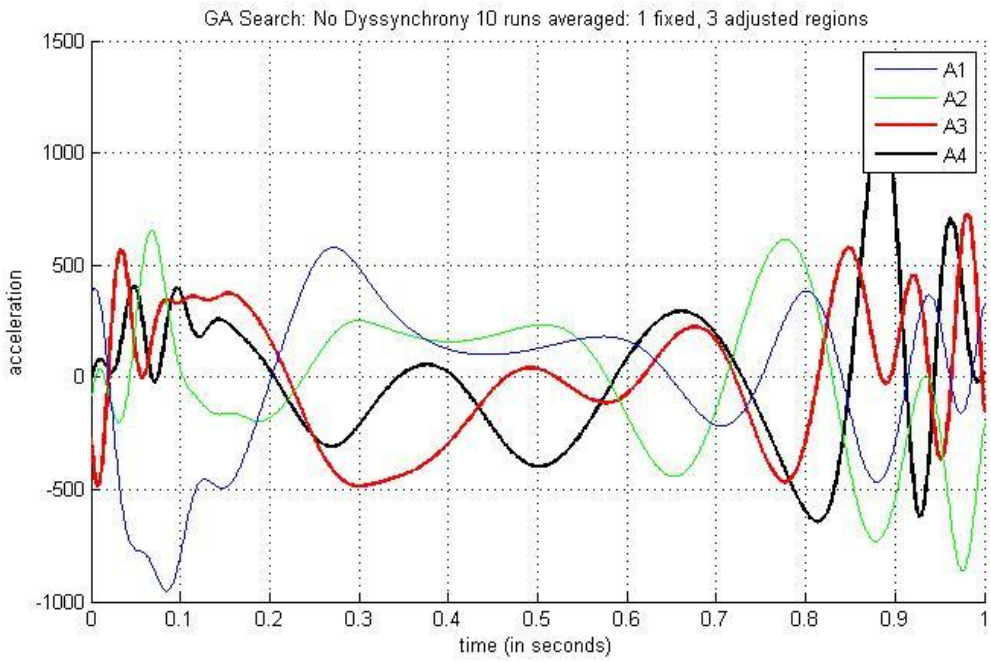

Figure A-33: No Dyssynchrony (1, 3) Acceleration for GA Search 10 Runs Avg, Mutation Rate = 0.05 (5%)

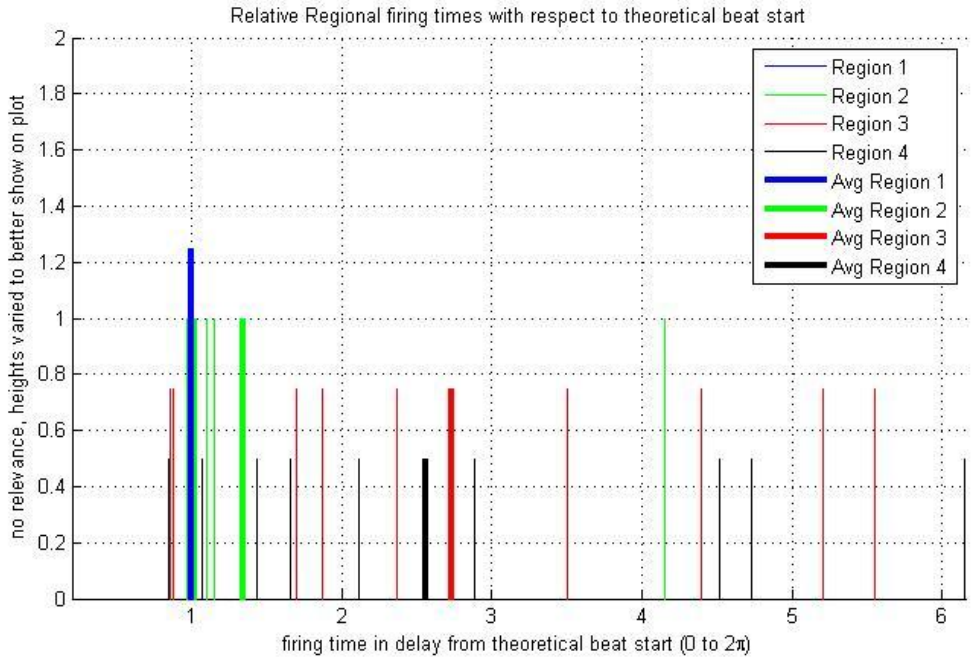

Figure A-34: No Dyssynchrony (1,3) Timing for GA Search 10 Runs, Mutation Rate = 0.05 (5%)

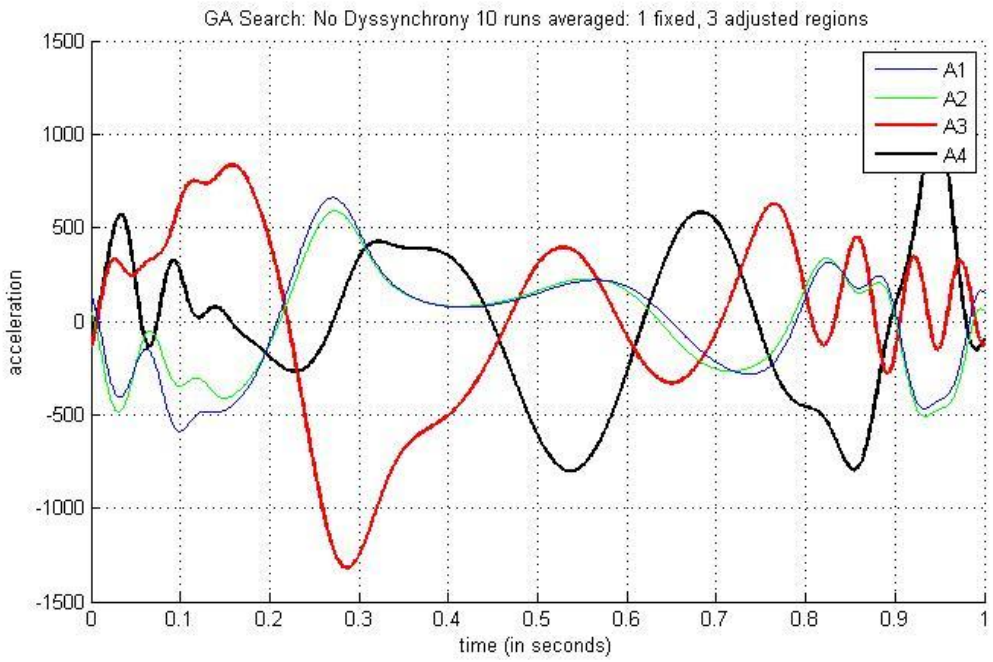

Figure A-35: No Dyssynchrony (1, 3) Acceleration for GA Search 10 Runs Avg, Mutation Rate = 0.10 (10%)

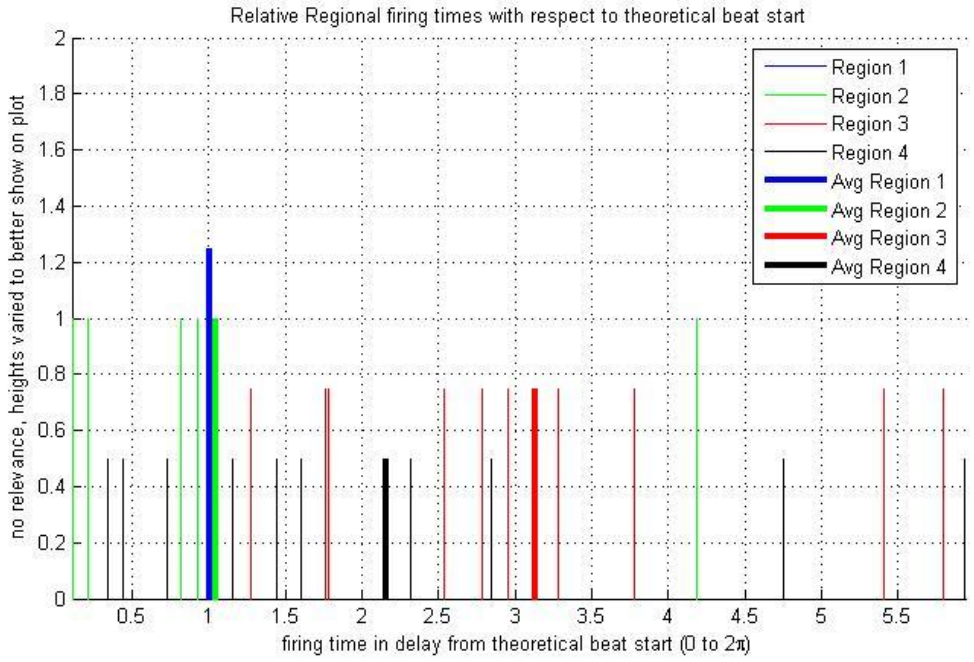

Figure A-36: No Dyssynchrony (1,3) Timing for GA Search 10 Runs, Mutation Rate = 0.10 (10%)

APPENDIX B. PLOTS OF VARIOUS EXHAUSTIVE SEARCH VARIATIONS FOR CD MODEL

Note: Labeling convention for acceleration plots (x, y) where x is the number of fixed regions, and y is the number of adjusted regions.

B.1. No Dyssynchrony

Case 1: Scripted_Initialization_no_dyss.m parameters

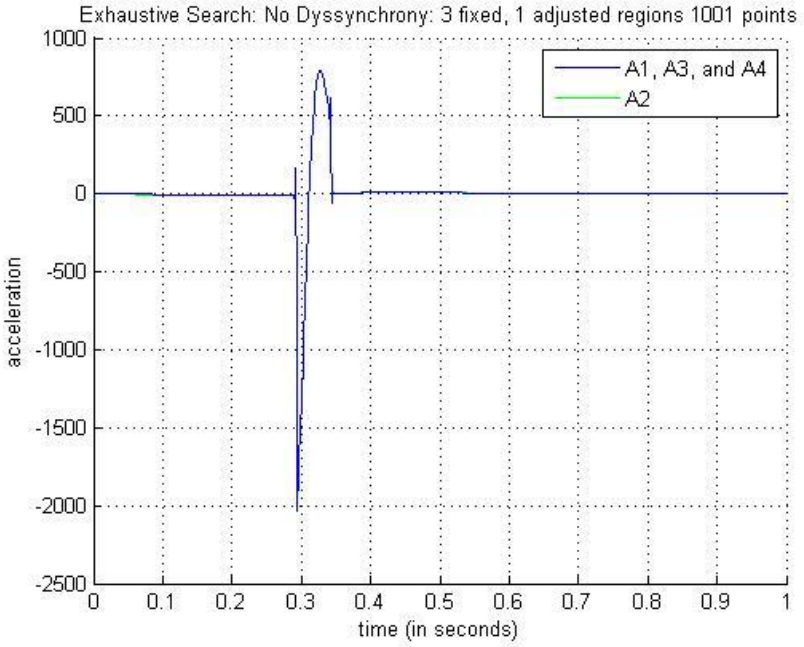

Figure B-1: No Dyssynchrony (3, 1) Acceleration for Exhaustive Search

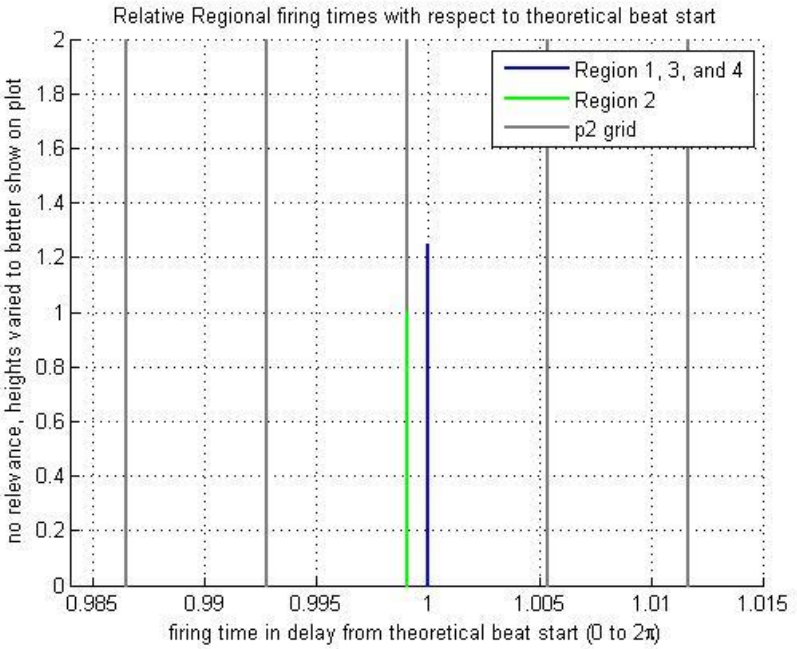

Figure B-2: No Dyssynchrony (3,1) Timing for Exhaustive Search

Case 2: Scripted_Initialization_no_dyss.m parameters

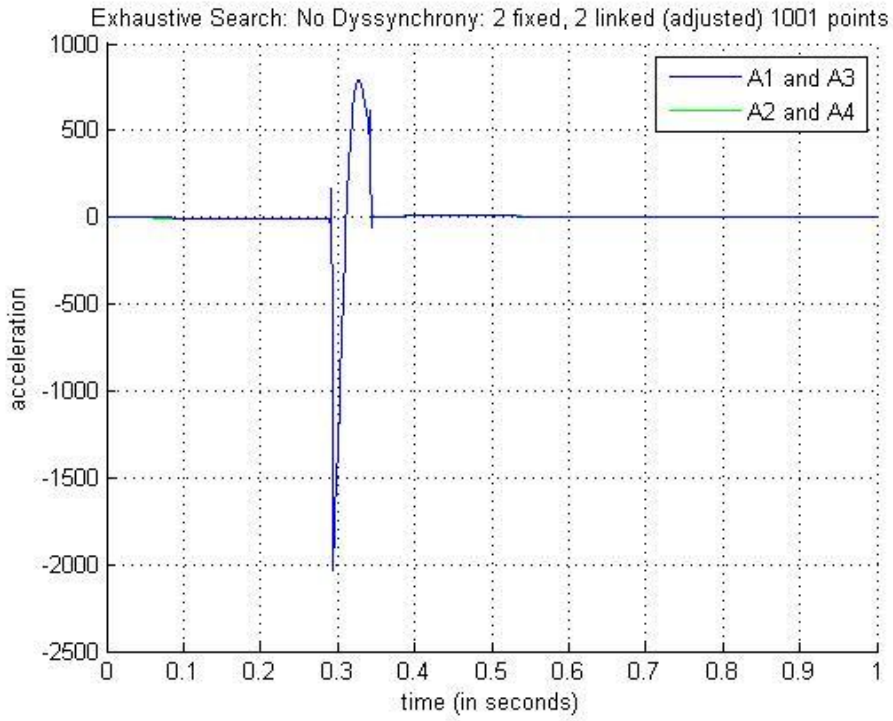

Figure B-3: No Dyssynchrony (2, 2) Acceleration for Exhaustive Search

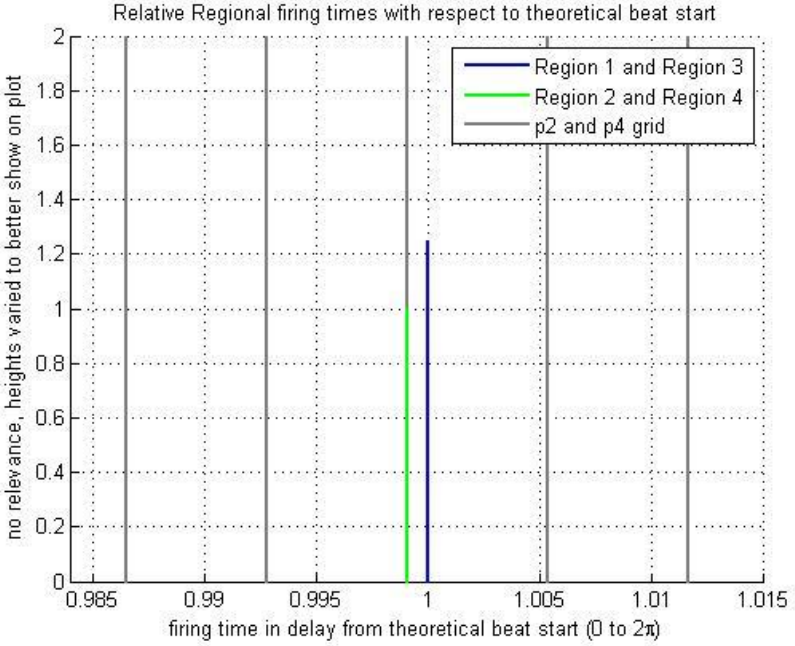

Figure B-4: No Dyssynchrony (2,2) Timing for Exhaustive Search

Case 3: Scripted_Initialization_no_dyss.m parameters

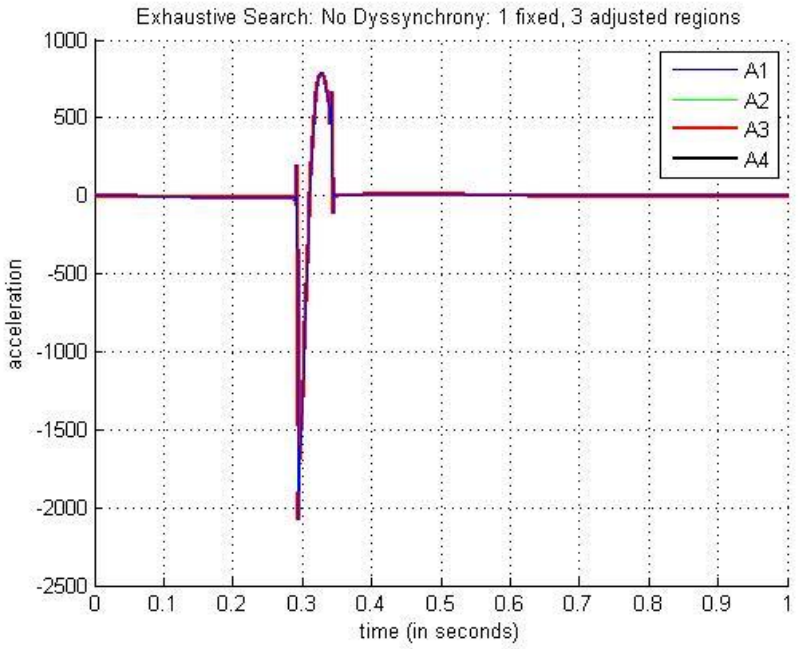

Figure B-5: No Dyssynchrony (1, 3) Acceleration for Exhaustive Search

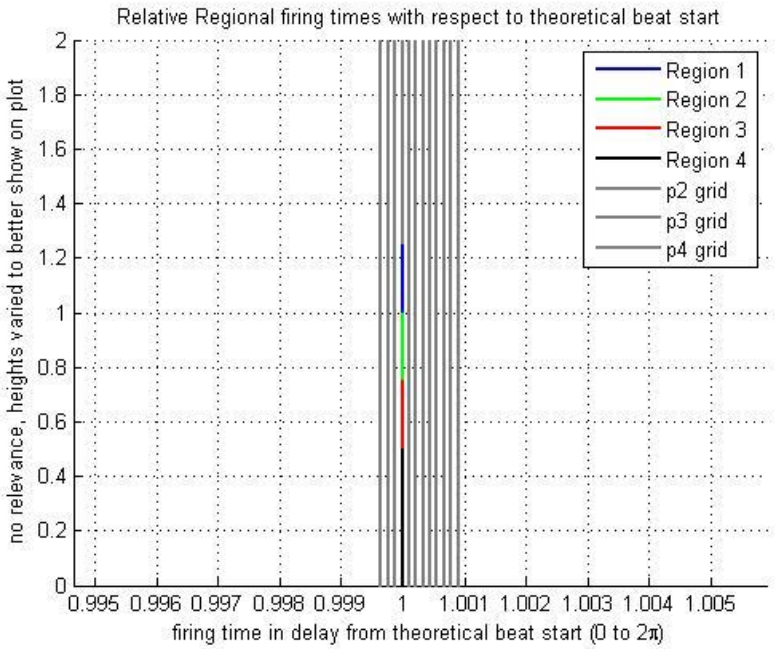

Figure B-6: No Dyssynchrony (1, 3) Acceleration for Exhaustive Search

B.2. Resistance Dyssynchrony

Case 1: Scripted_Initialization_r2_dyss_0_015.m parameters

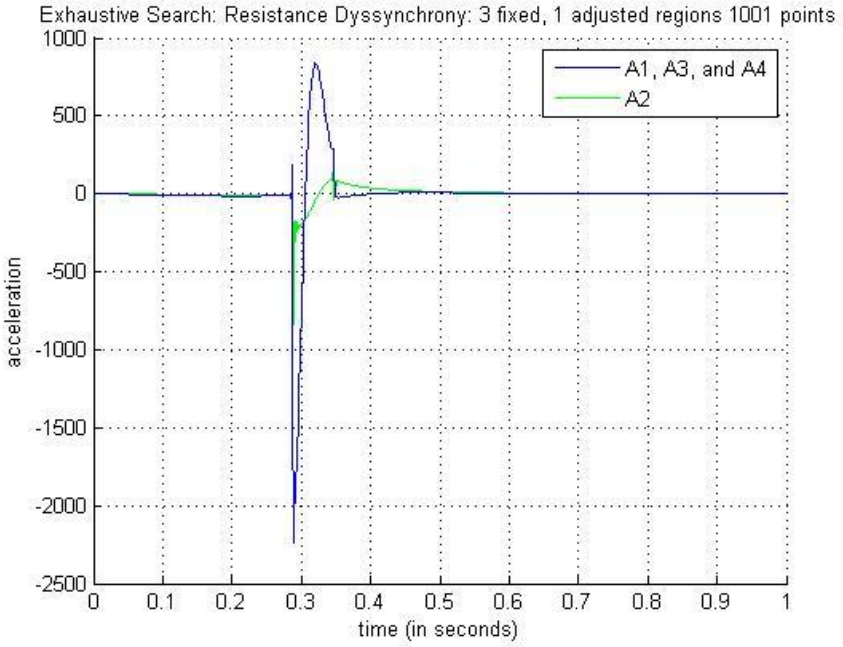

Figure B-7: Resistance Dyssynchrony (3, 1) Acceleration for Exhaustive Search

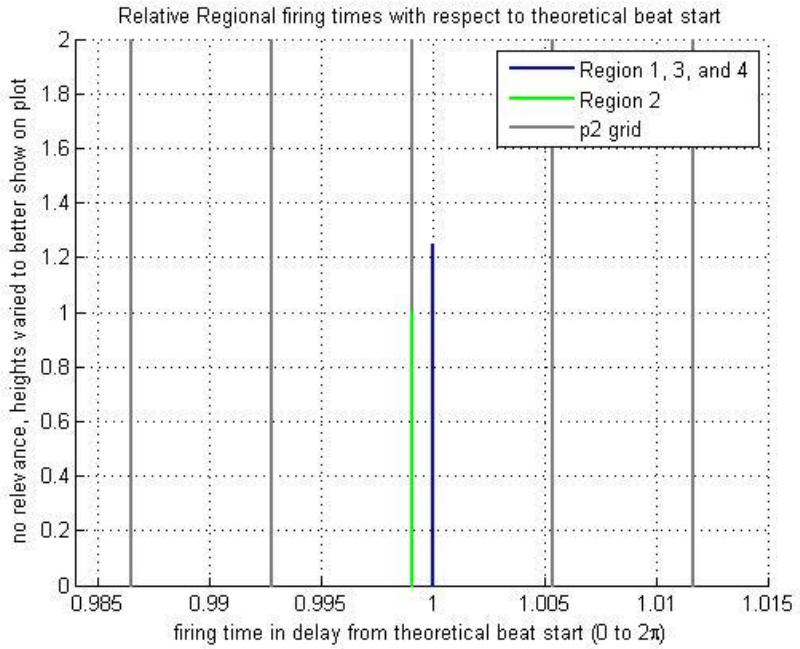

Figure B-8: Resistance Dyssynchrony (3,1) Timing for Exhaustive Search

Case 2: Scripted_Initialization_r2_dyss_0_015.m parameters

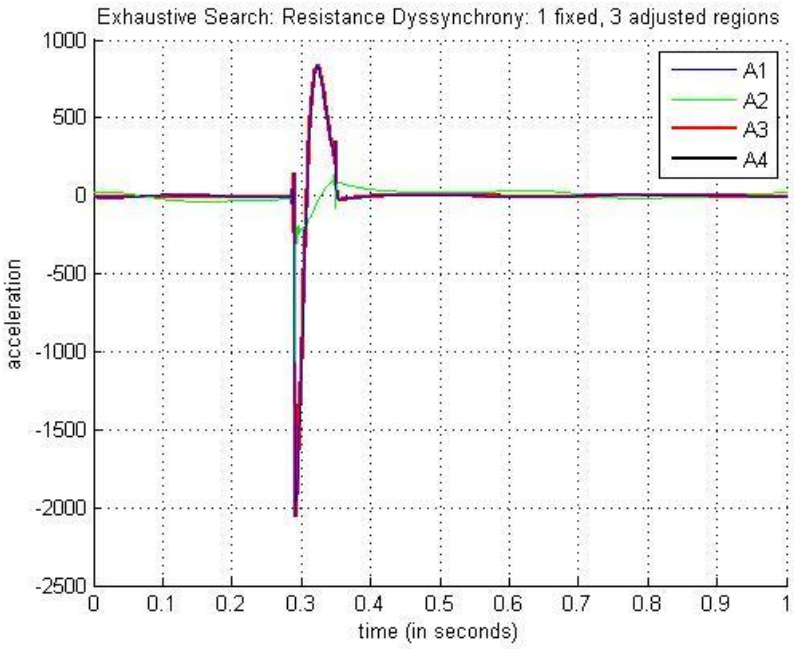

Figure B-9: Resistance Dyssynchrony (1, 3) Acceleration for Exhaustive Search

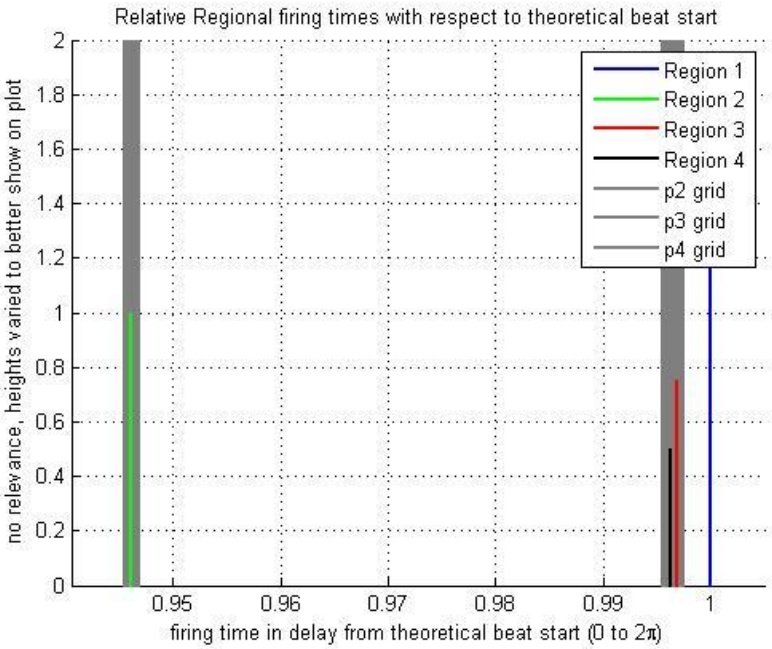

Figure B-10: Resistance Dyssynchrony (1,3) Timing for Exhaustive Search

B.3. Mass Dyssynchrony

Case 1: Scripted_Initialization_m2_dyss_0_01.m parameters

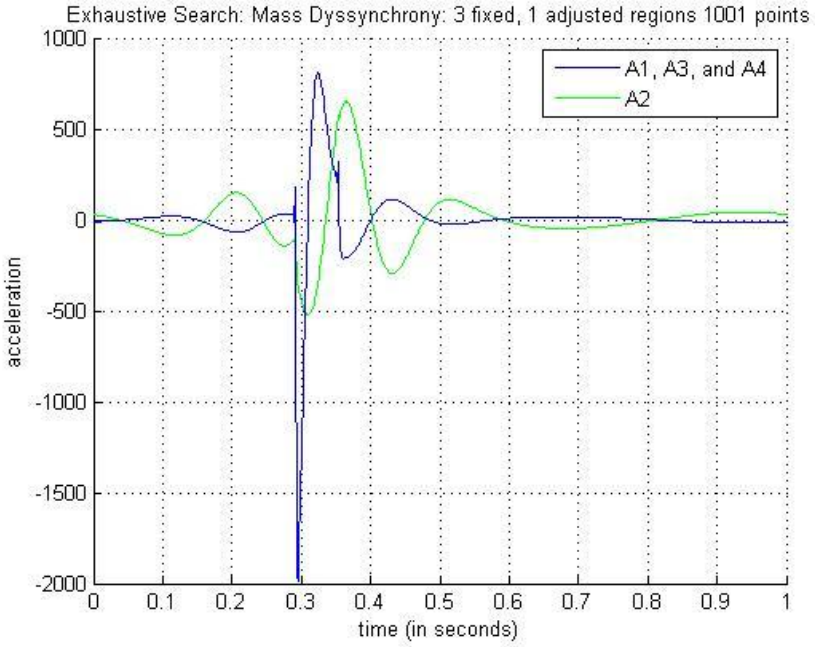

Figure B-11: Mass Dyssynchrony (3, 1) Acceleration for Exhaustive Search

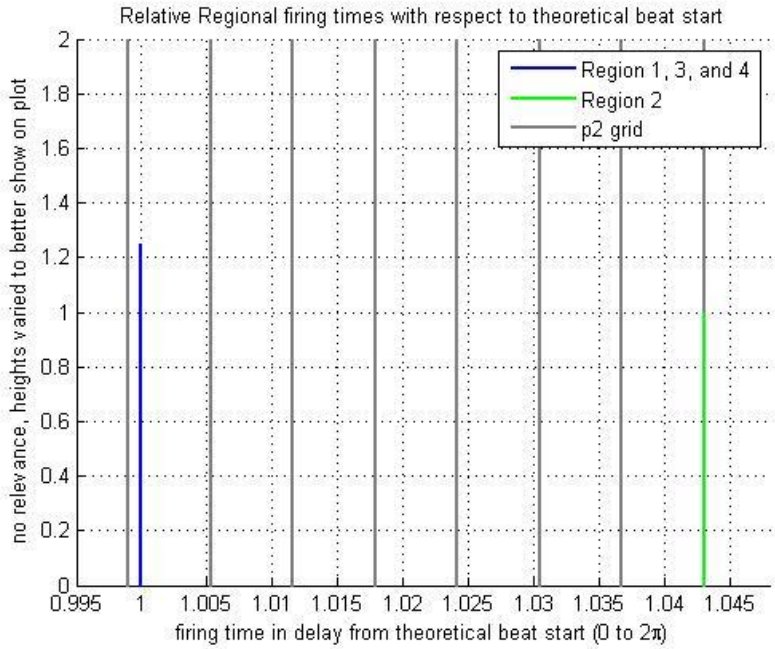

Figure B-12: Mass Dyssynchrony (3,1) Timing for Exhaustive Search

Case 2: Scripted_Initialization_m2_dyss_0_01.m parameters

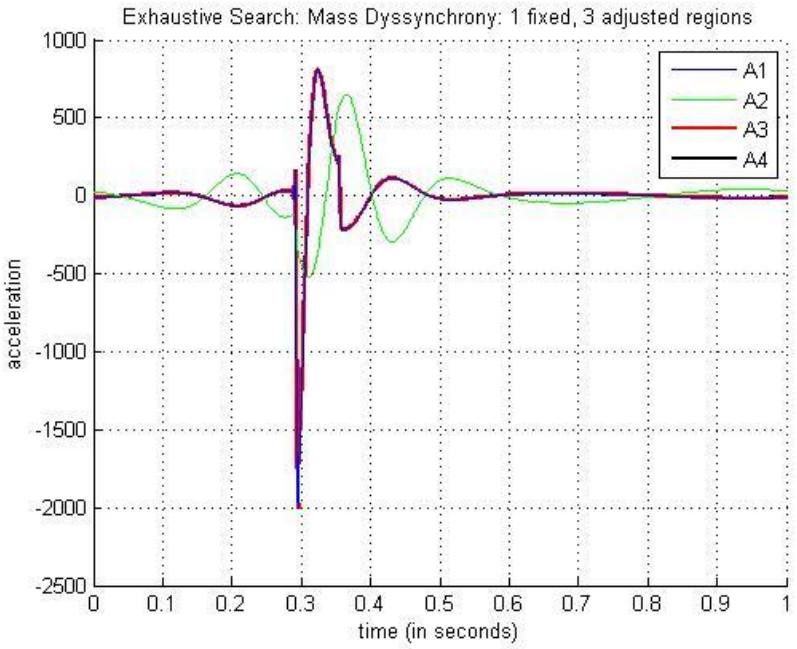

Figure B-13: Mass Dyssynchrony (1, 3) Acceleration for Exhaustive Search

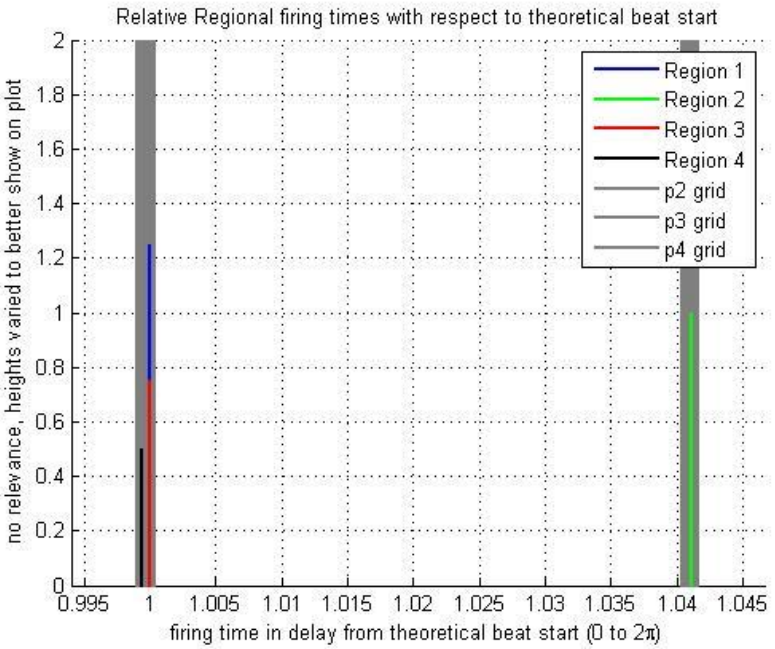

Figure B-14: Mass Dyssynchrony (1,3) Timing for Exhaustive Search

B.4. Minimum Elastance Dyssynchrony

Case 1: Scripted_Initialization_min_elas2_dyss_4.m parameters

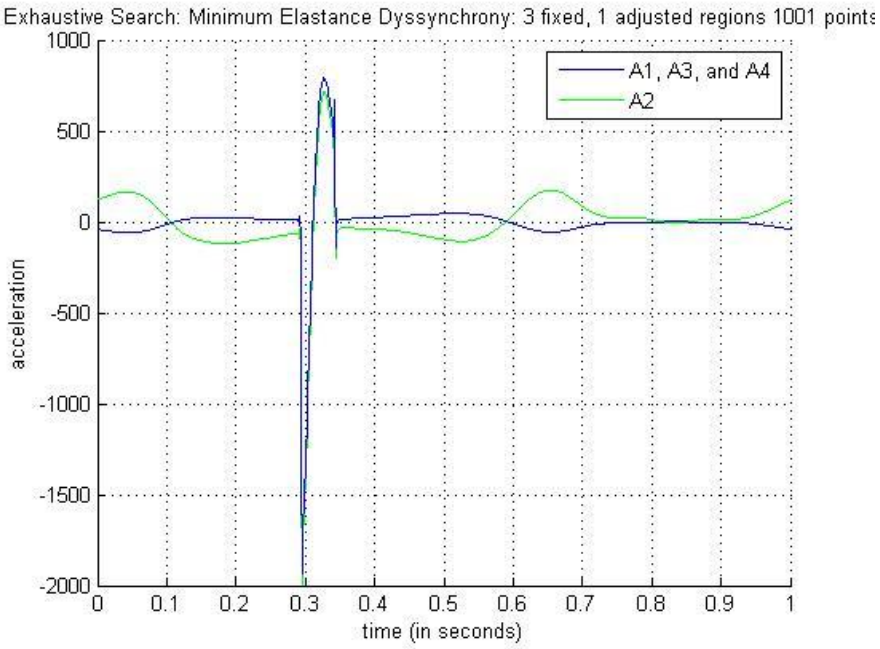

Figure B-15: Min Elastance Dyssynchrony (3, 1) Acceleration for Exhaustive Search

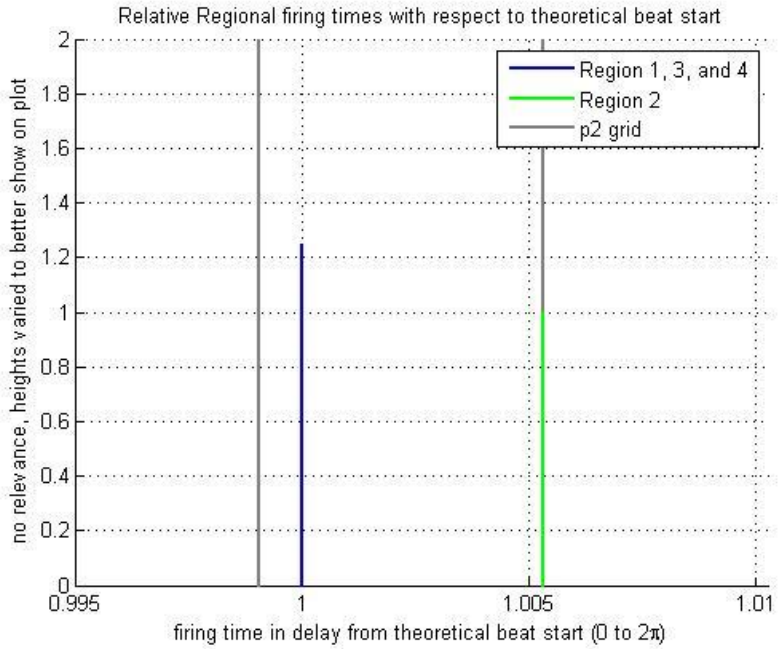

Figure B-16: Min Elastance Dyssynchrony (3,1) Timing for Exhaustive Search

Case 2: Scripted_Initialization_min_elas2_dyss_4.m parameters

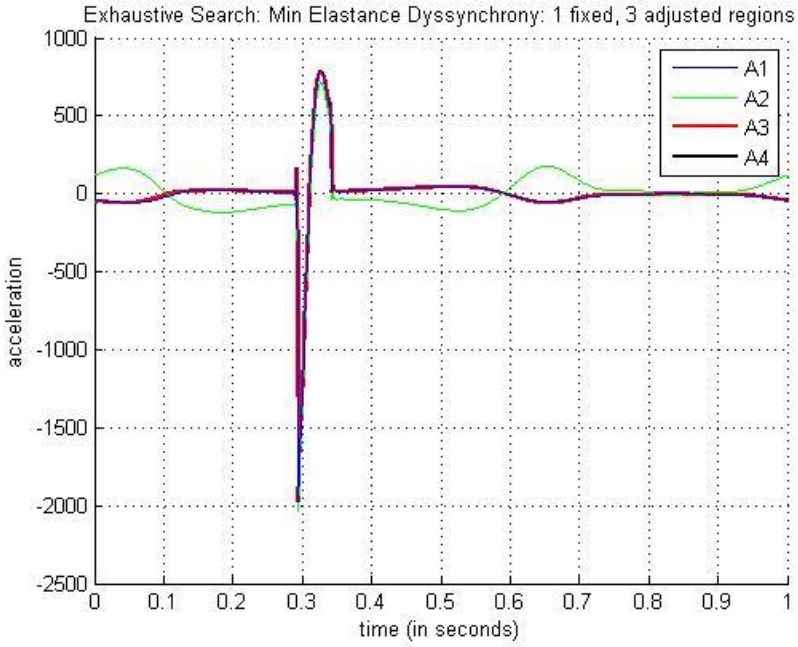

Figure B-17: Min Elastance Dyssynchrony (1, 3) Acceleration for Exhaustive Search

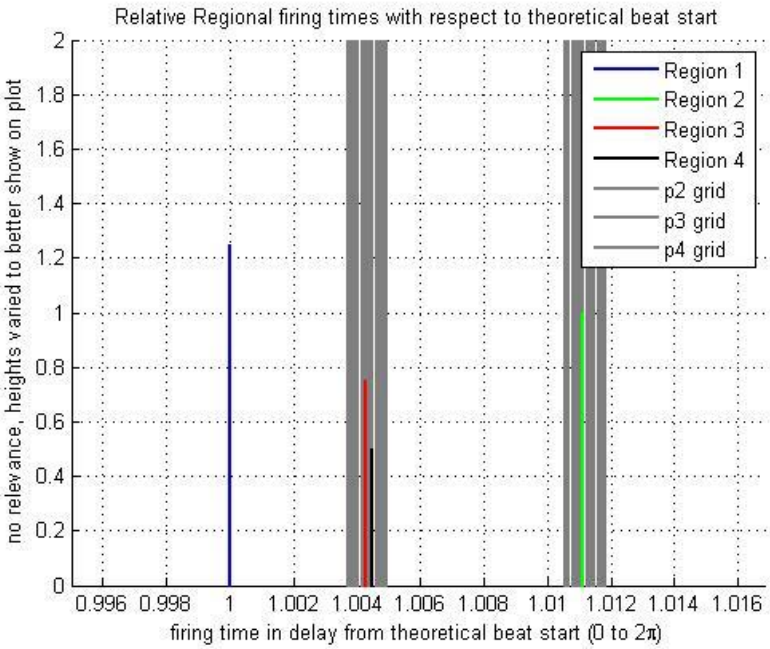

Figure B-18: Min Elastance Dyssynchrony (1,3) Timing for Exhaustive Search

B.5. Maximum Elastance Dyssynchrony

Case 1: Scripted_Initialization_max_elas2_dyss_40.m parameters

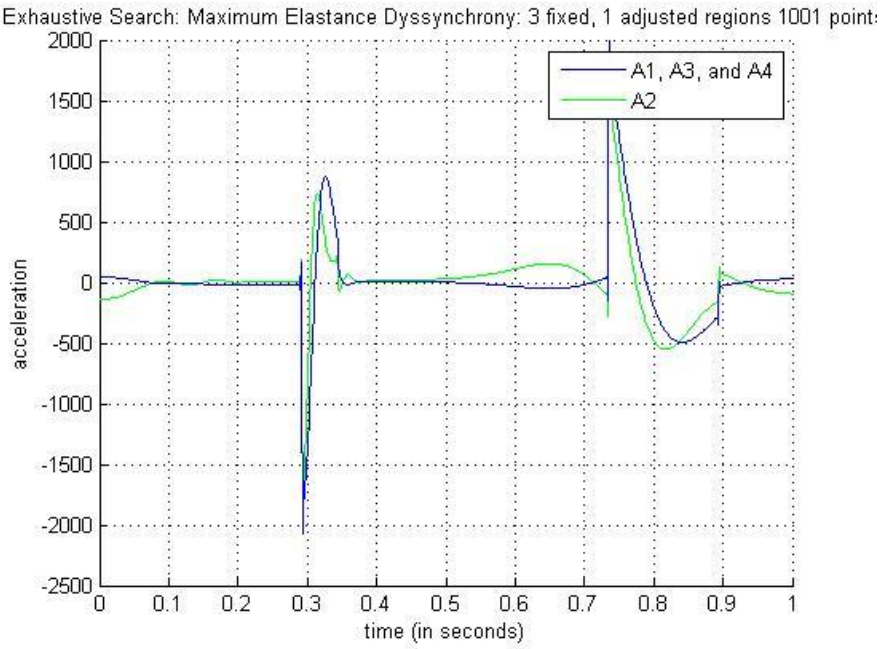

Figure B-19: Max Elastance Dyssynchrony (3, 1) Acceleration for Exhaustive Search

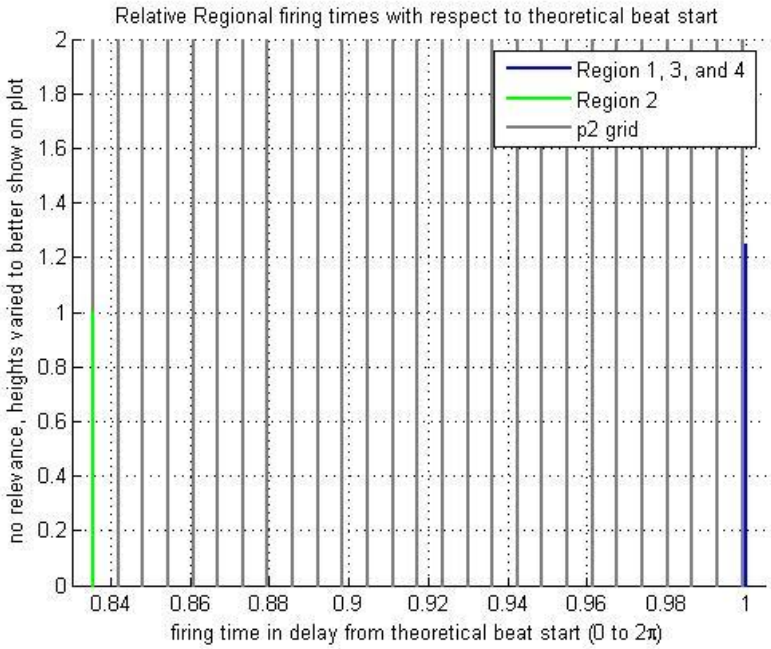

Figure B-20: Max Elastance Dyssynchrony (3,1) Timing for Exhaustive Search

Case 2: Scripted_Initialization_max_elas2_dyss_40.m parameters

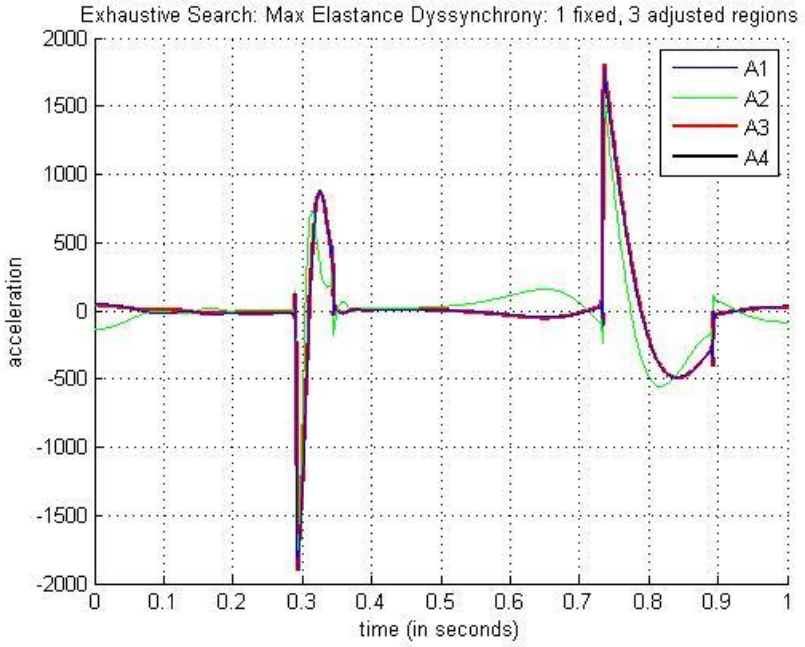

Figure B-21: Max Elastance Dyssynchrony (1, 3) Acceleration for Exhaustive Search

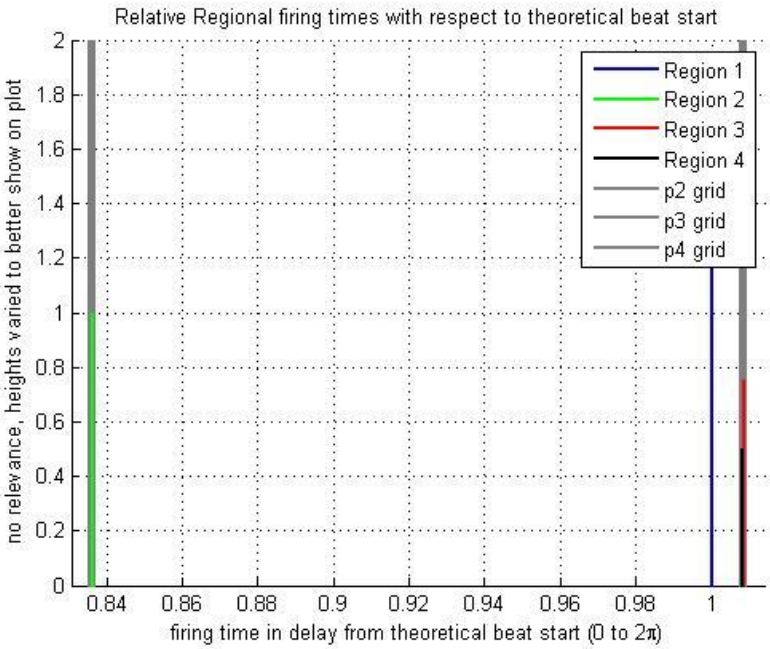

Figure B-22: Max Elastance Dyssynchrony (1,3) Timing for Exhaustive Search

B.6. Combined Dyssynchrony (all dyssynchrony)

Region 2: Scripted_Initialization_all_dyss_region2.m parameters

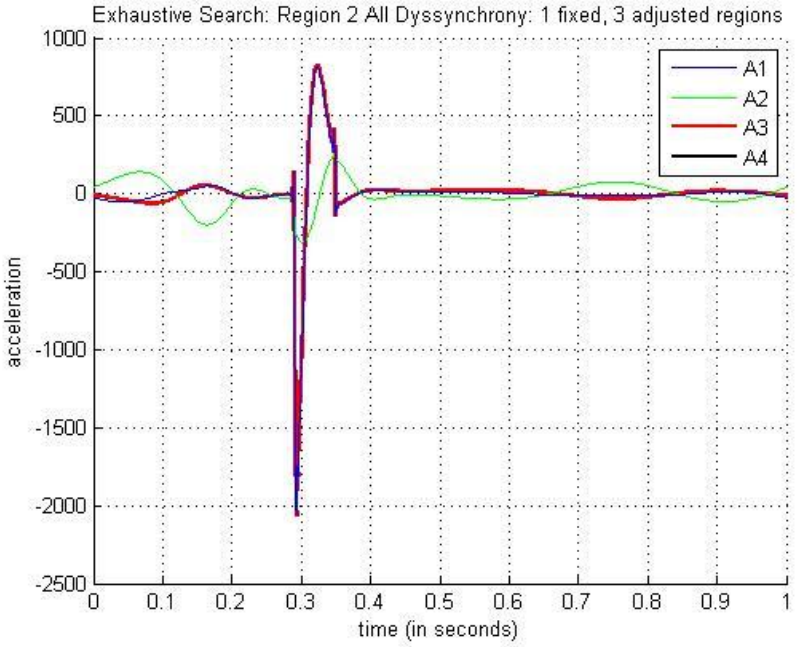

Figure B-23: All Dyssynchrony Region 2 (1, 3) Acceleration for Exhaustive Search

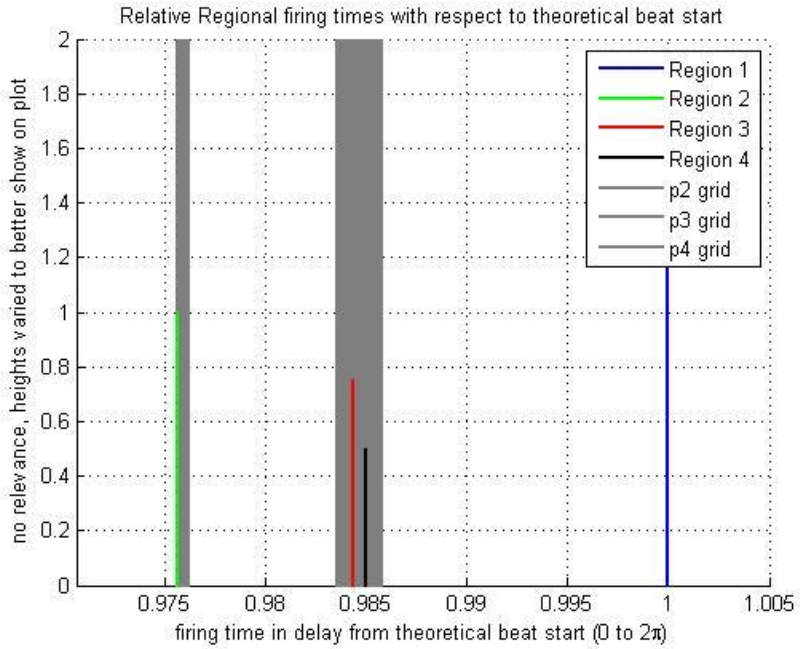

Figure B-24: All Dyssynchrony Region 2 (3,1) Timing for Exhaustive Search

Regions 2, 3, and 4: Scripted_Initialization_all_dyss_region234.m parameters

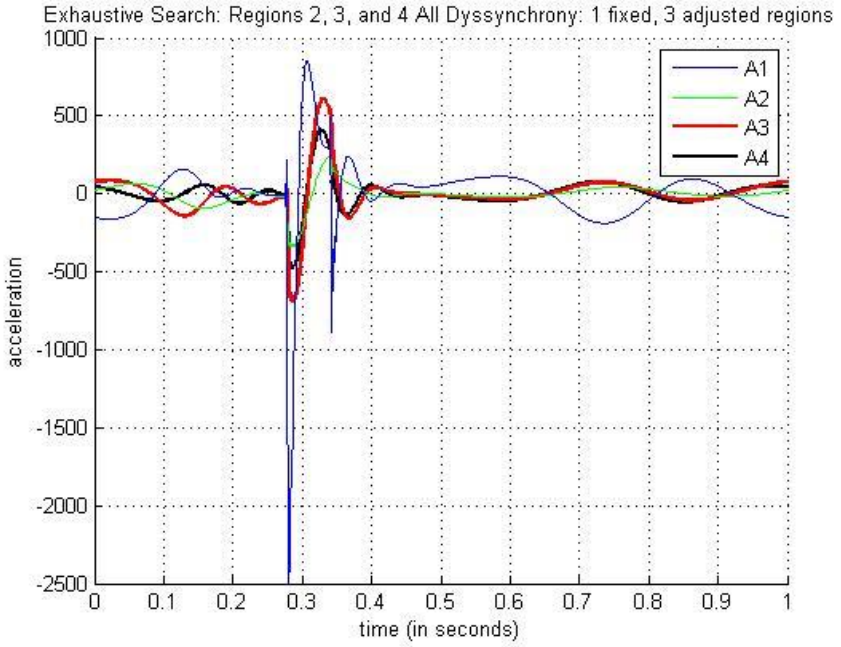

Figure B-25: All Dyssynchrony Regions 2, 3, and 4 (1, 3) Acceleration for Exhaustive Search

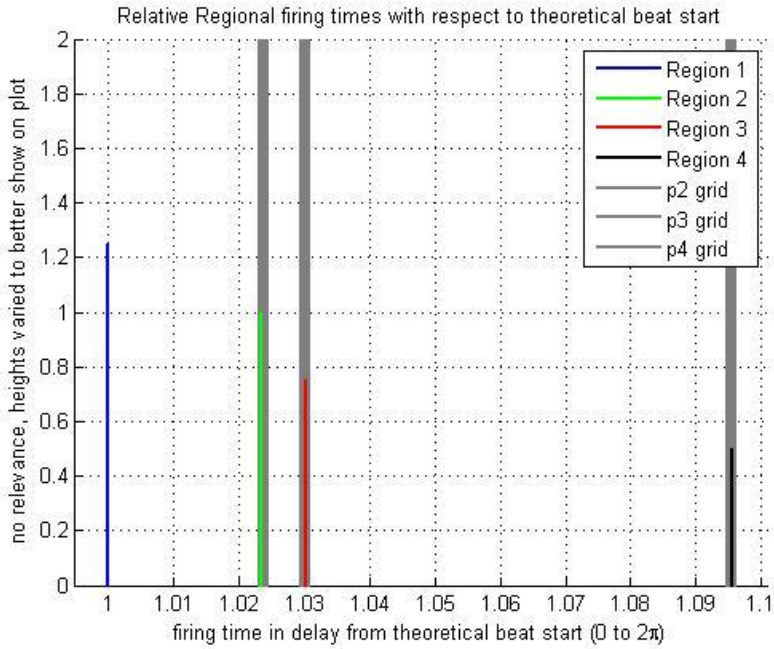

Figure B-26: All Dyssynchrony Regions 2, 3, and 4 (3,1) Timing for Exhaustive Search

APPENDIX C. PLOTS OF NO TIMING ADJUSTMENT FOR CD MODEL (ALL TIMES ARE EQUAL)

Note: Labeling convention for acceleration plots (x, y) where x is the number of fixed regions, and y is the number of adjusted regions.

C.1. No Dyssynchrony

Case 1: Scripted_Initialization_no_dyss.m parameters

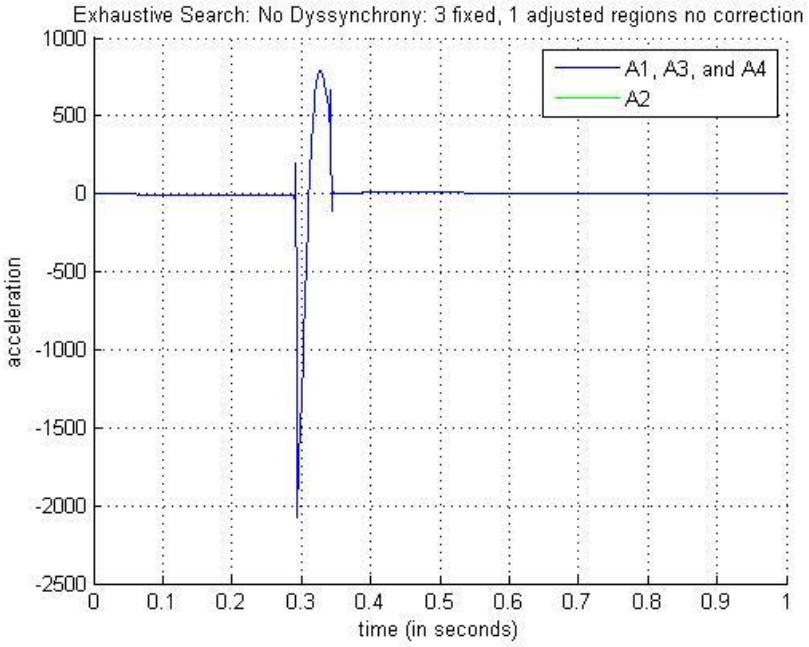

Figure C-1: No Dyssynchrony (3, 1) Acceleration for No Adjustment

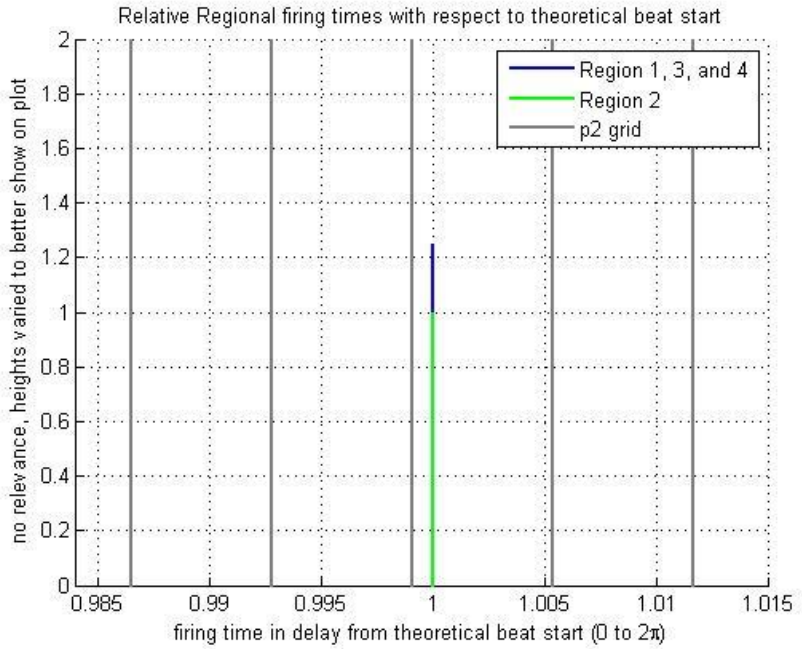

Figure C-2: No Dyssynchrony (3,1) Timing for No Adjustment

Case 2: Scripted_Initialization_no_dyss.m parameters

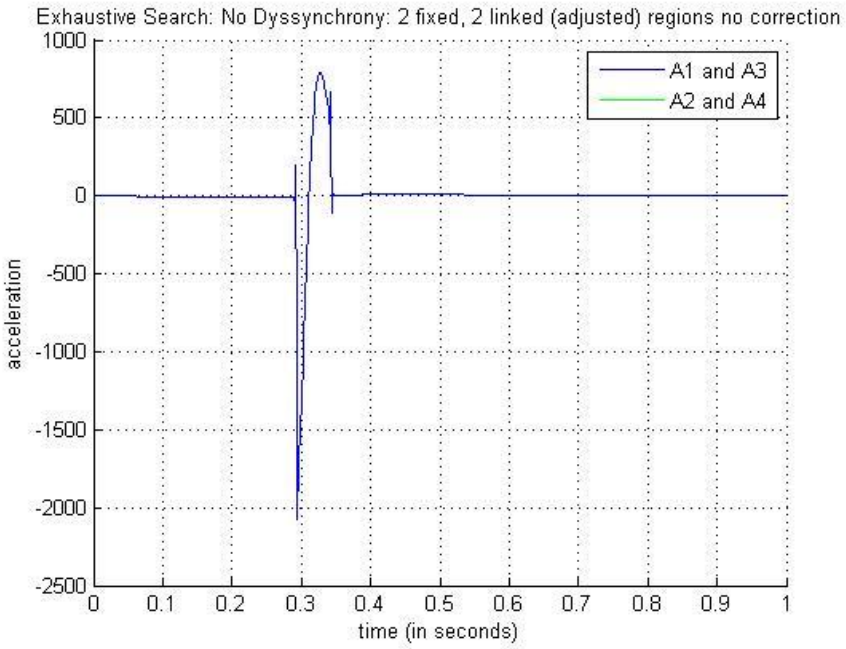

Figure C-3: No Dyssynchrony (2, 2) Acceleration for No Adjustment

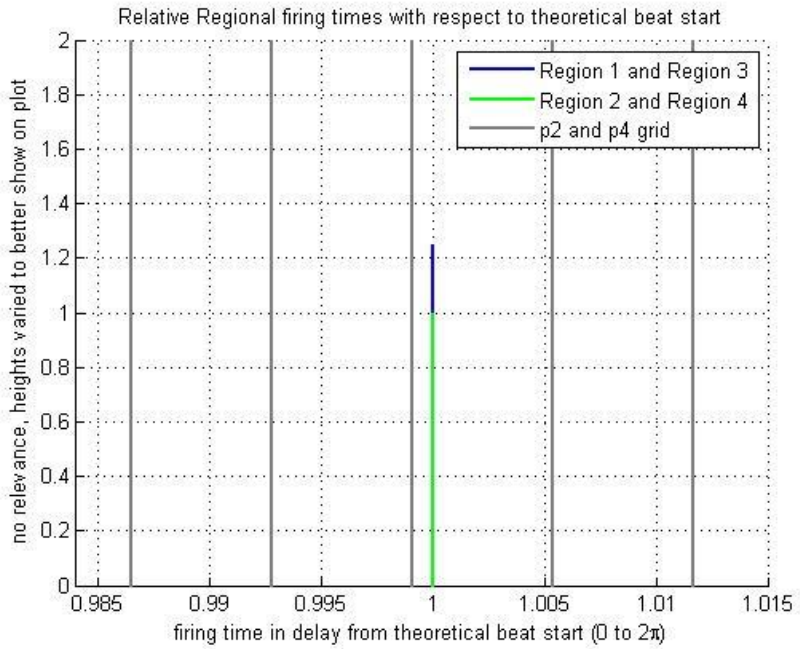

Figure C-4: No Dyssynchrony (2,2) Timing for No Adjustment

Case 3: Scripted_Initialization_no_dyss.m parameters

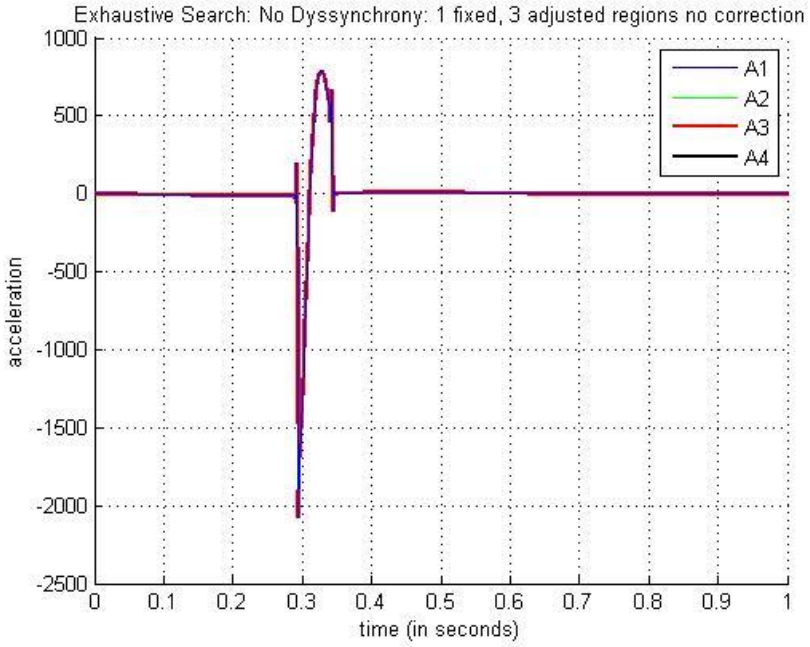

Figure C-5: No Dyssynchrony (1, 3) Acceleration for No Adjustment

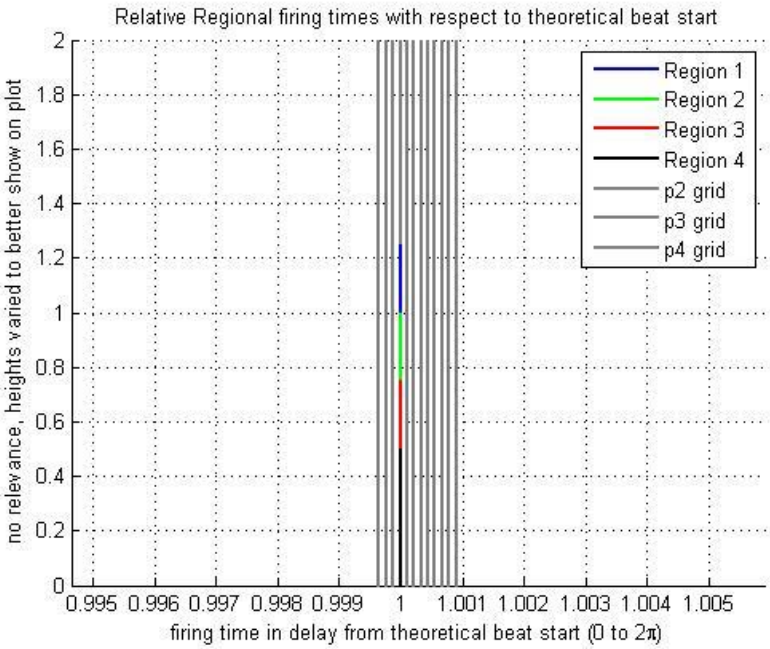

Figure C-6: No Dyssynchrony (1,3) Timing for No Adjustment

C.2. Resistance Dyssynchrony

Case 1: Scripted_Initialization_r2_dyss_0_015.m parameters

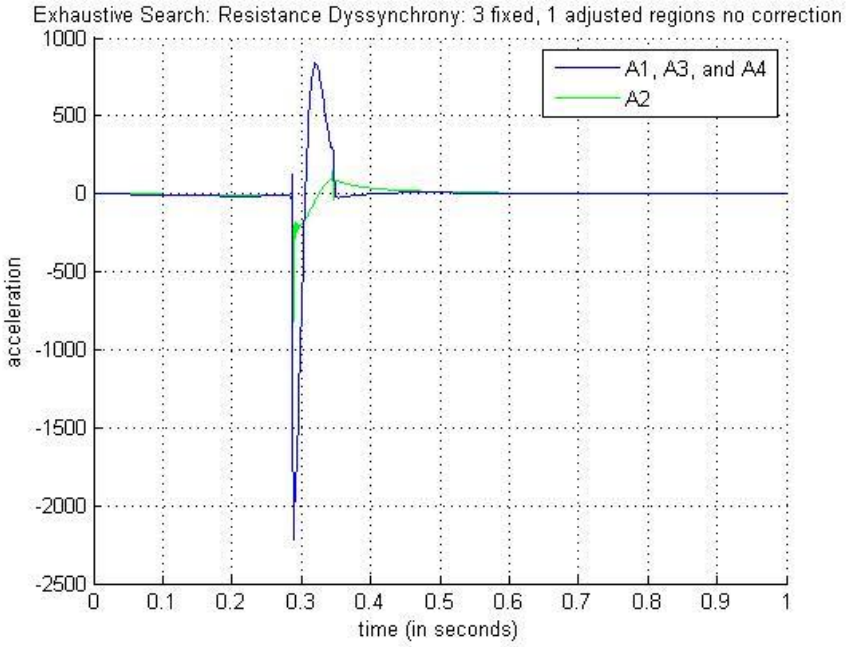

Figure C-7: Resistance Dyssynchrony (3, 1) Acceleration for No Adjustment

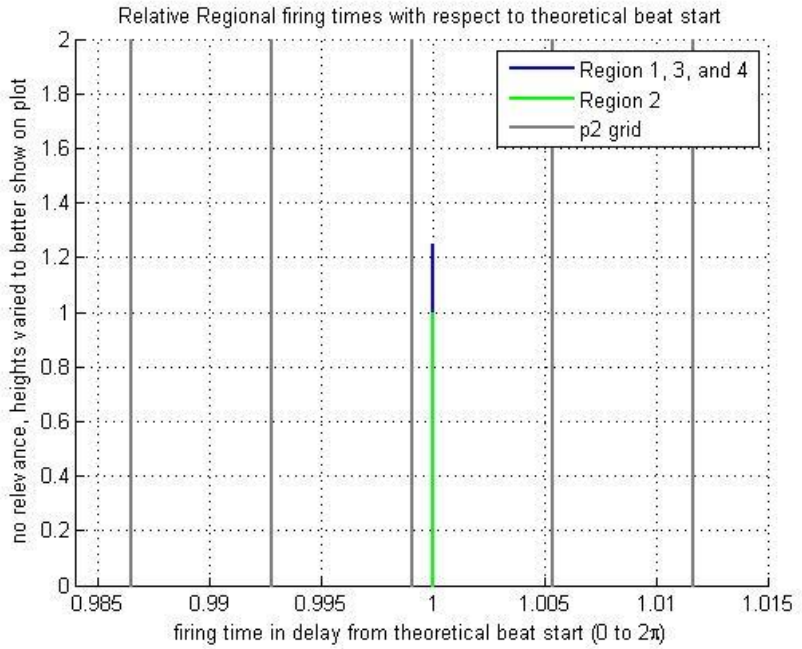

Figure C-8: Resistance Dyssynchrony (3,1) Timing for No Adjustment

Case 2: Scripted_Initialization_r2_dyss_0_015.m parameters

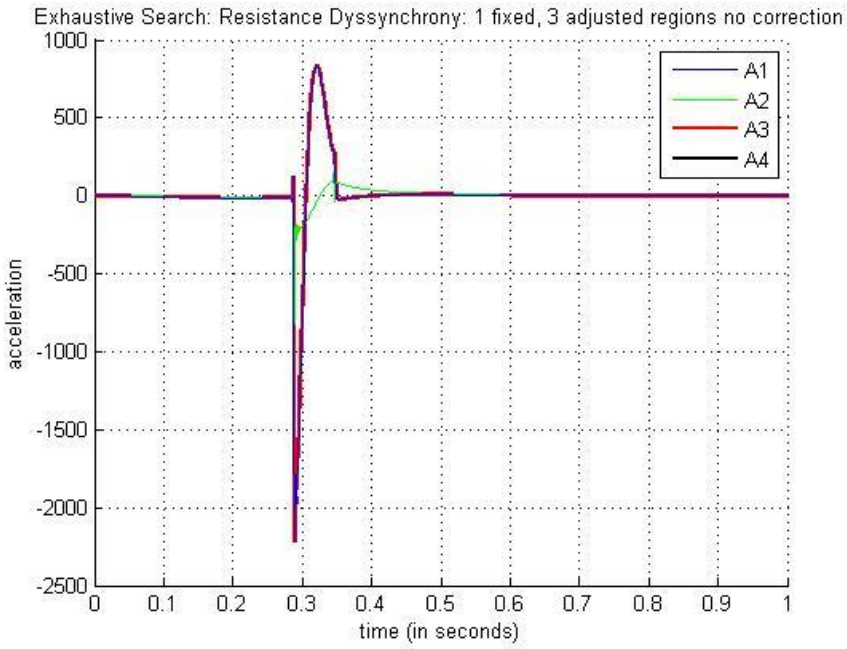

Figure C-9: Resistance Dyssynchrony (1, 3) Acceleration for No Adjustment

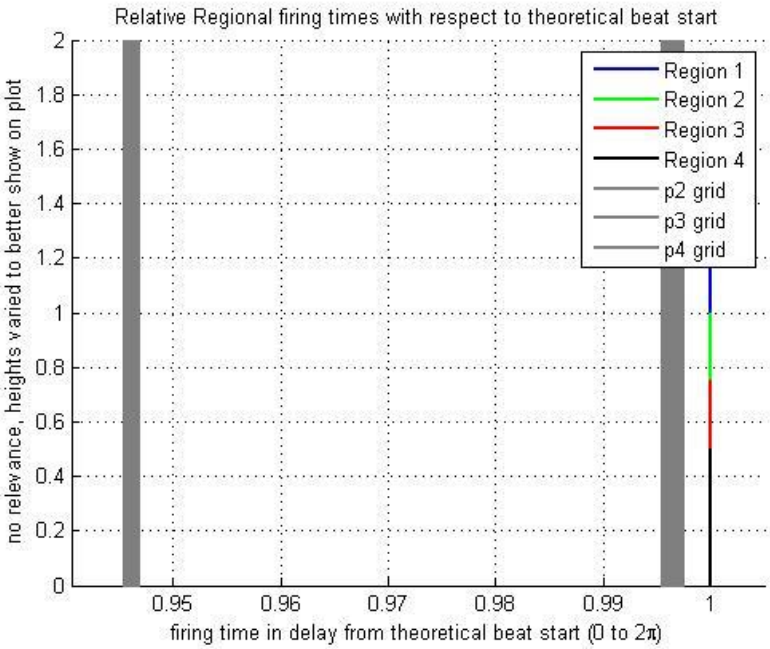

Figure C-10: Resistance Dyssynchrony (1,3) Timing for No Adjustment

C.3. Mass Dyssynchrony

Case 1: Scripted_Initialization_m2_dyss_0_01.m parameters

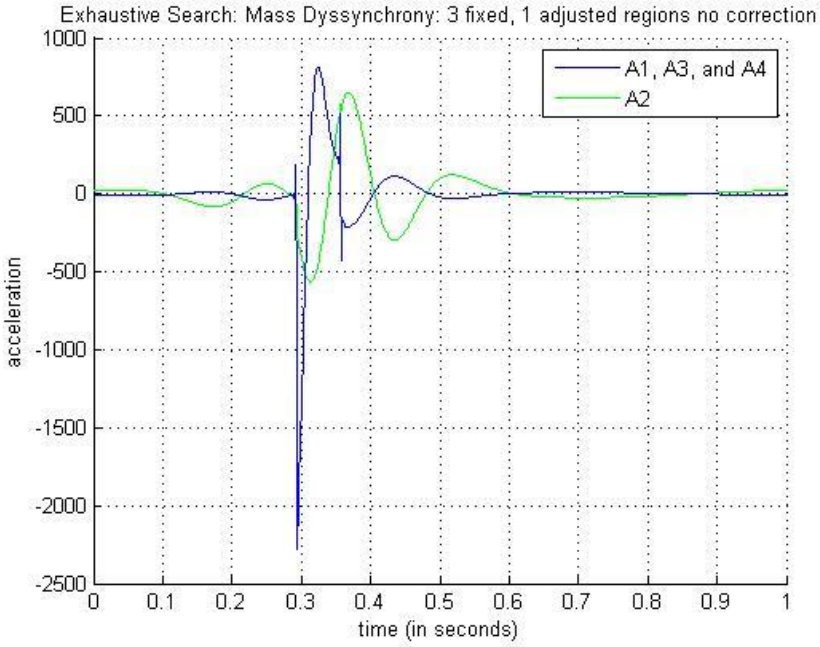

Figure C-11: Mass Dyssynchrony (3, 1) Acceleration for No Adjustment

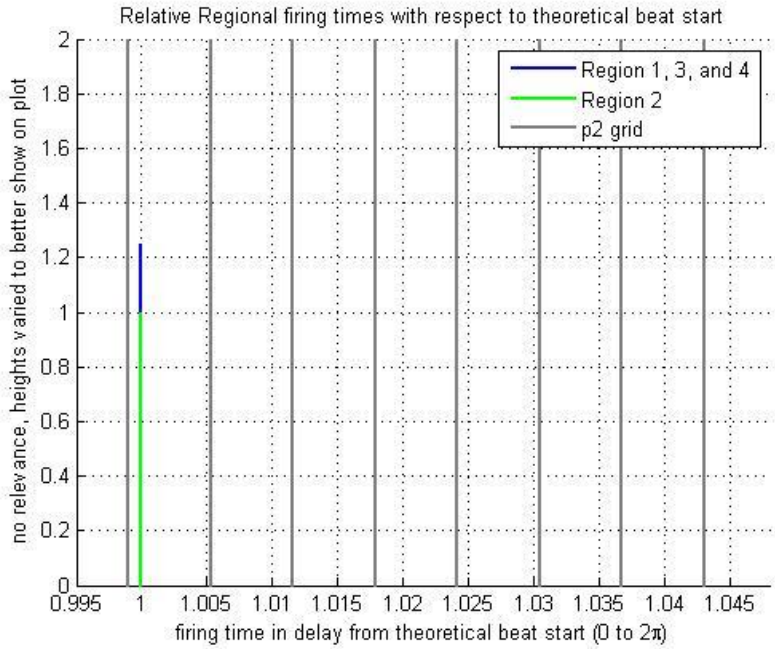

Figure C-12: Mass Dyssynchrony (3,1) Timing for No Adjustment

Case 2: Scripted_Initialization_m2_dyss_0_01.m parameters

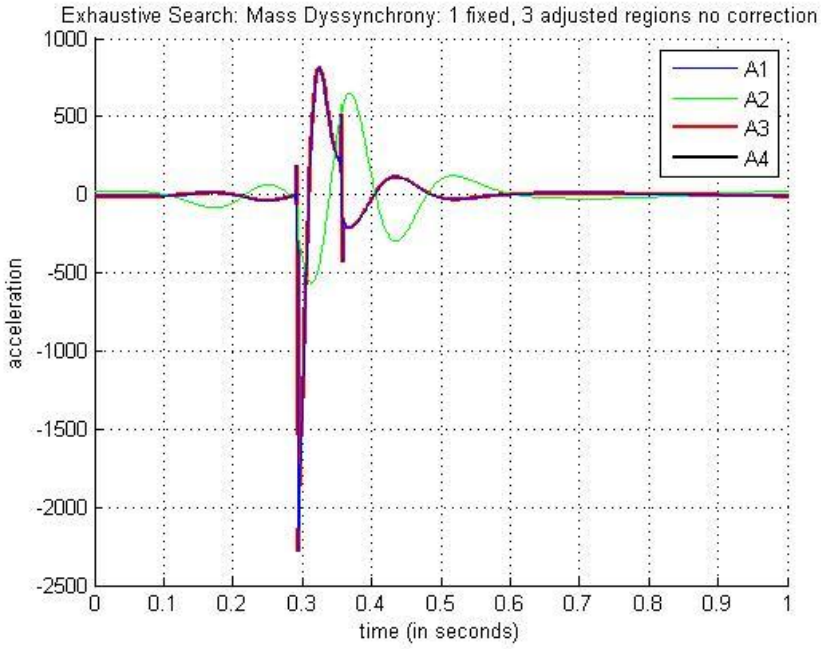

Figure C-13: Mass Dyssynchrony (1, 3) Acceleration for No Adjustment

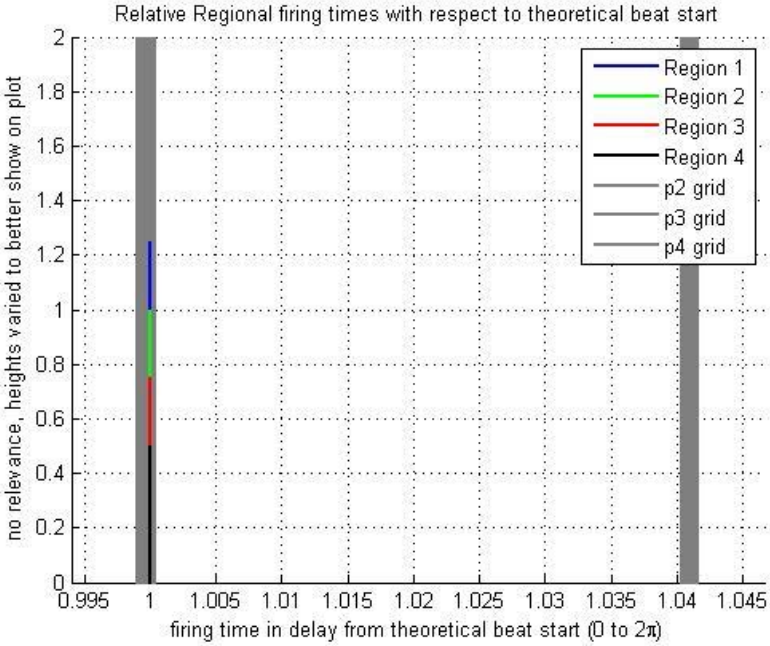

Figure C-14: Mass Dyssynchrony (1,3) Timing for No Adjustment

C.4. Minimum Elastance Dyssynchrony

Case 1: Scripted_Initialization_min_elas2_dyss_4.m parameters

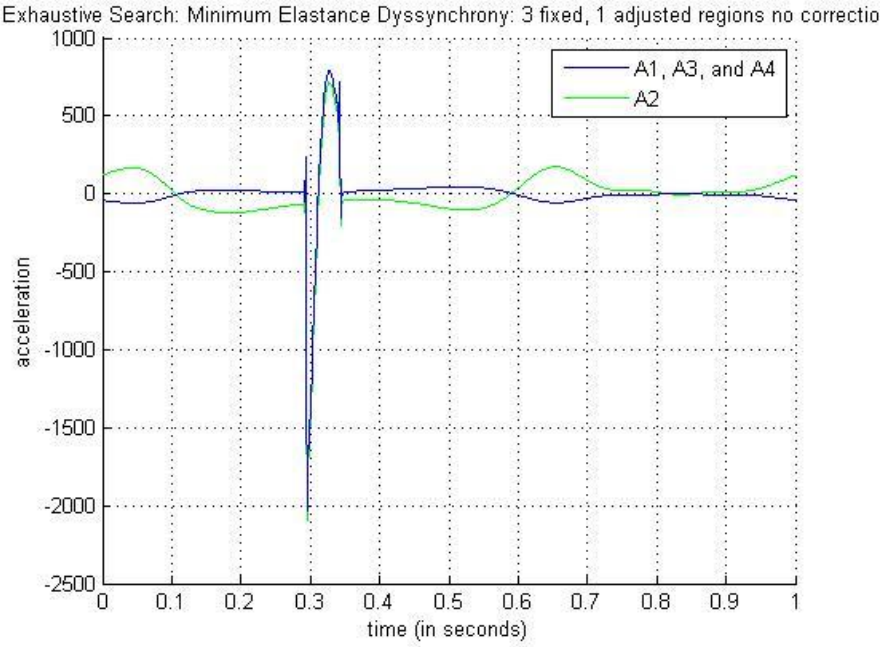

Figure C-15: Min Elastance Dyssynchrony (3, 1) Acceleration for No Adjustment

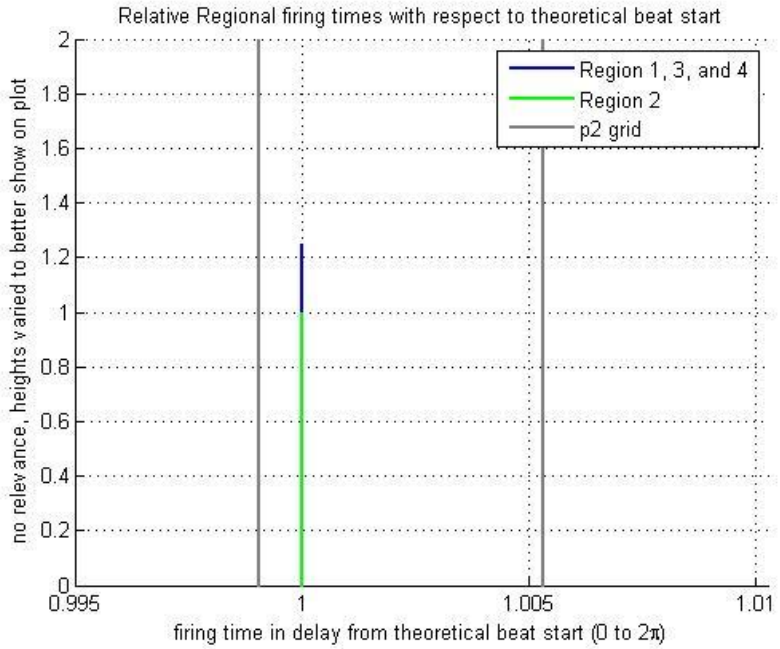

Figure C-16: Min Elastance Dyssynchrony (3,1) Timing for No Adjustment

Case 2: Scripted_Initialization_min_elas2_dyss_4.m parameters

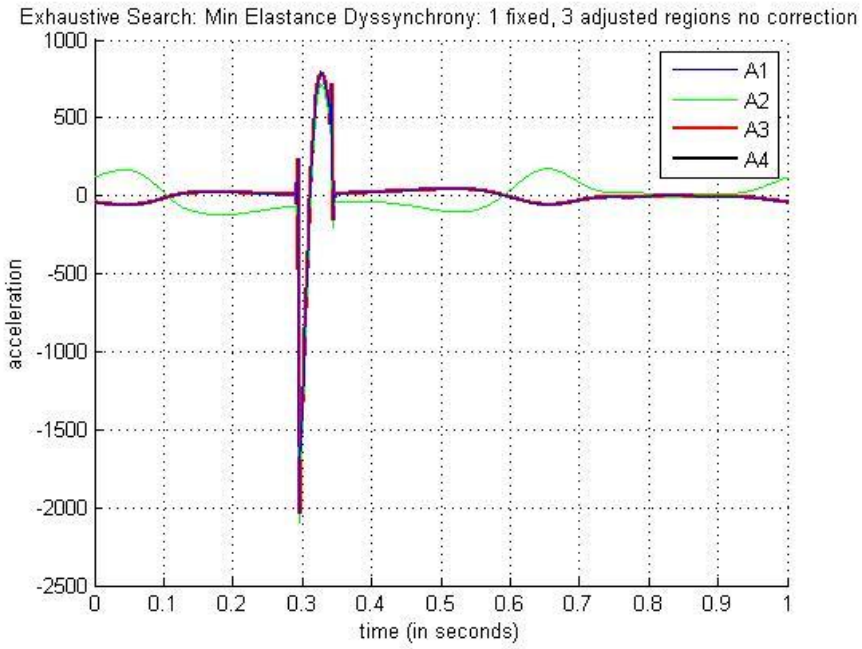

Figure C-17: Min Elastance Dyssynchrony (1, 3) Acceleration for No Adjustment

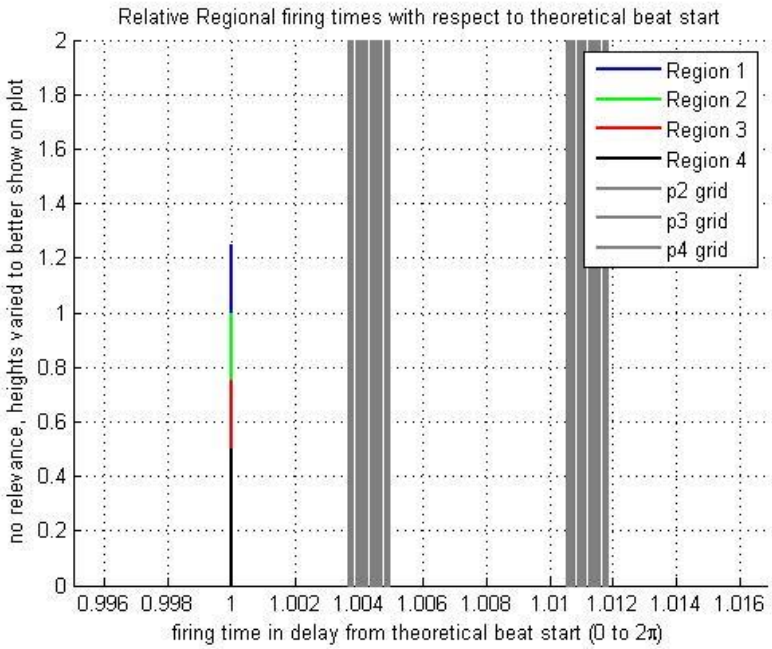

Figure C-18: Min Elastance Dyssynchrony (1,3) Timing for No Adjustment

C.5. Maximum Elastance Dyssynchrony

Case 1: Scripted_Initialization_max_elas2_dyss_40.m parameters

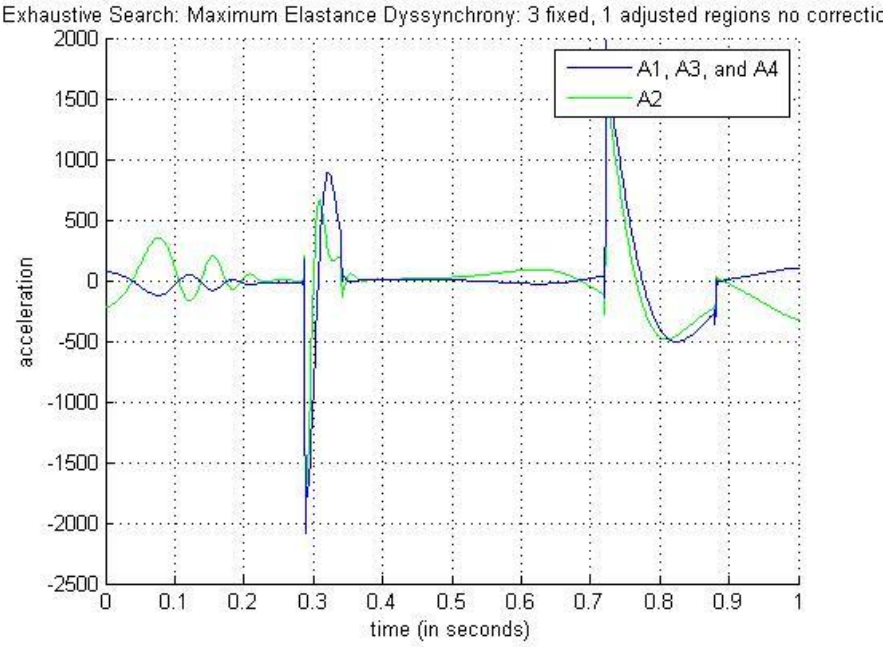

Figure C-19: Max Elastance Dyssynchrony (3, 1) Acceleration for No Adjustment

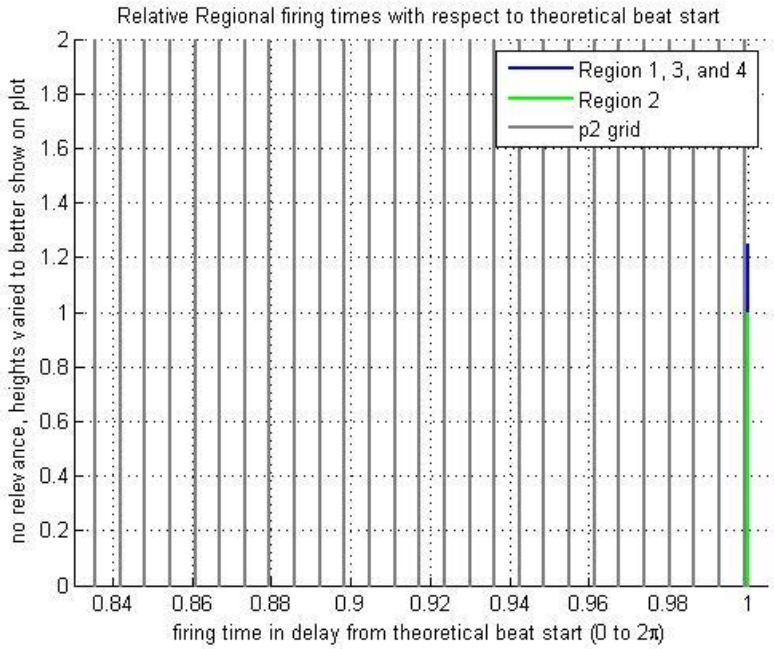

Figure C-20: Max Elastance Dyssynchrony (3,1) Timing for No Adjustment

Case 2: Scripted_Initialization_max_elas2_dyss_40.m parameters

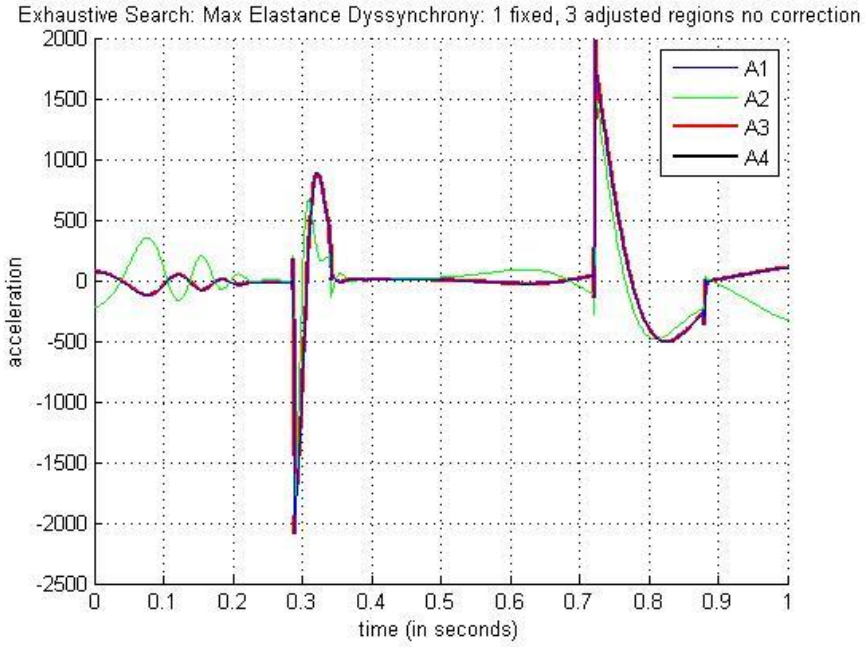

Figure C-21: Max Elastance Dyssynchrony (1, 3) Acceleration for No Adjustment

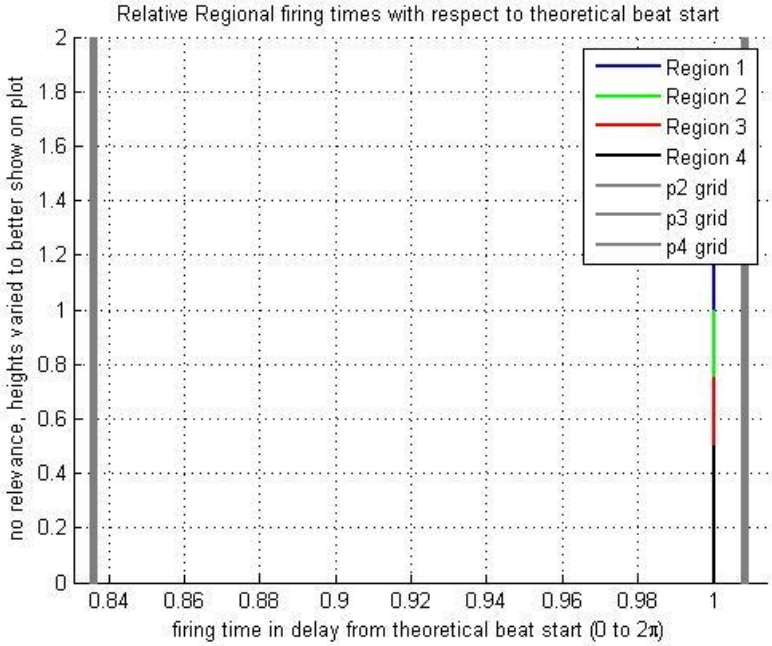

Figure C-22: Max Elastance Dyssynchrony (1,3) Timing for No Adjustment

C.6. Combined Dyssynchrony (all dyssynchrony)

Region 2: Scripted_Initialization_all_dyss_region2.m parameters

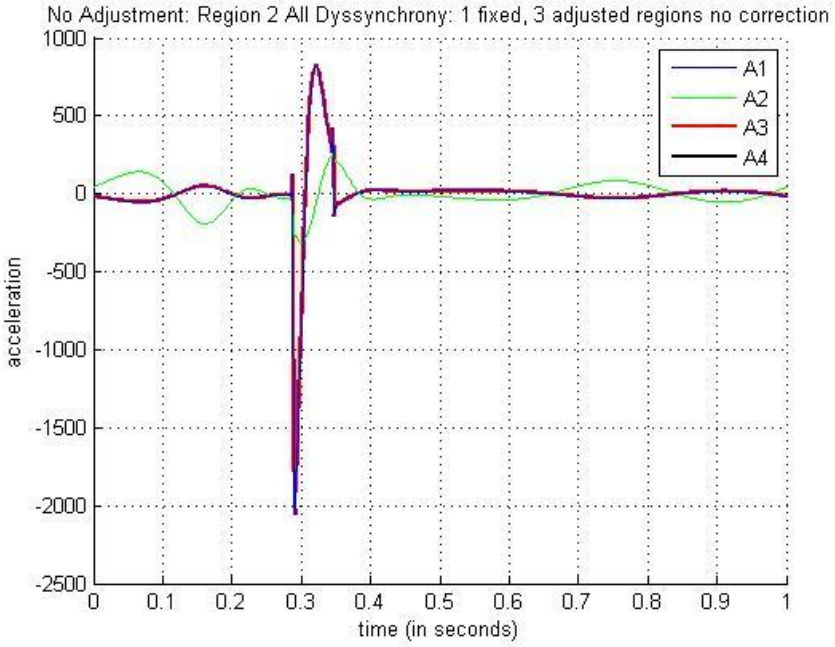

Figure C-23: All Dyssynchrony Region 2 (1, 3) Acceleration for No Adjustment

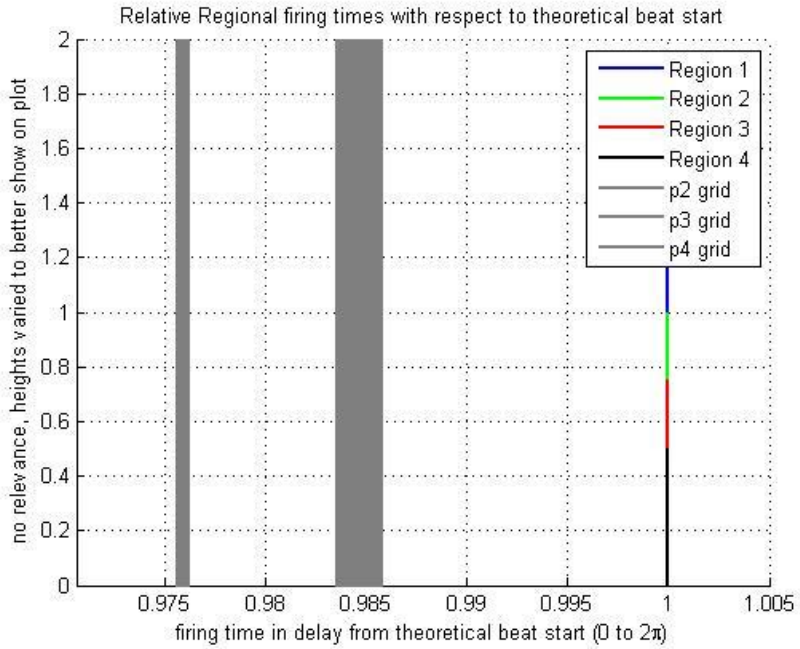

Figure C-24: All Dyssynchrony Region 2 (3,1) Timing for No Adjustment

Regions 2, 3, and 4: Scripted_Initialization_all_dyss_region234.m parameters

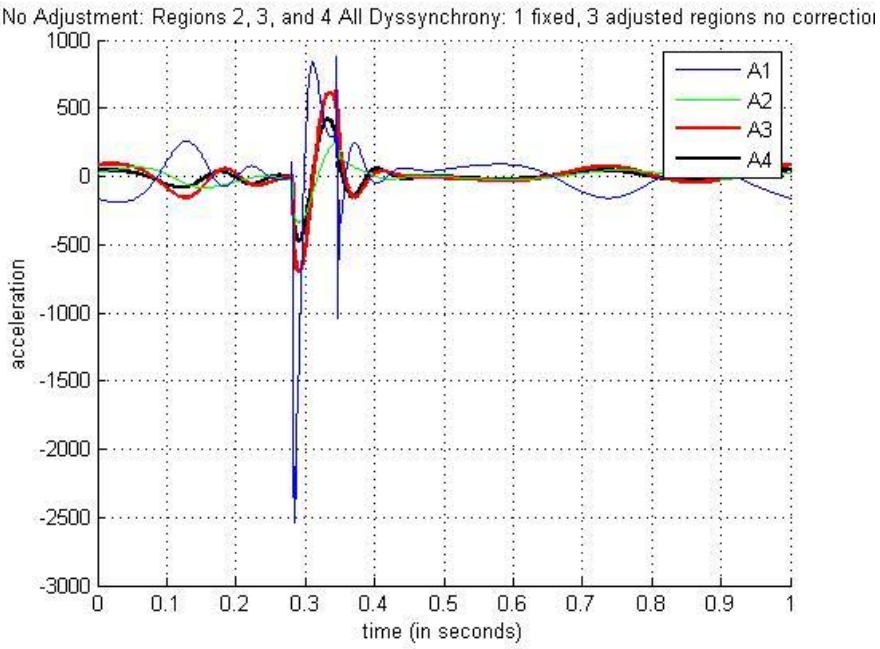

Figure C-25: All Dyssynchrony Regions 2, 3, and 4 (1, 3) Acceleration for No Adjustment

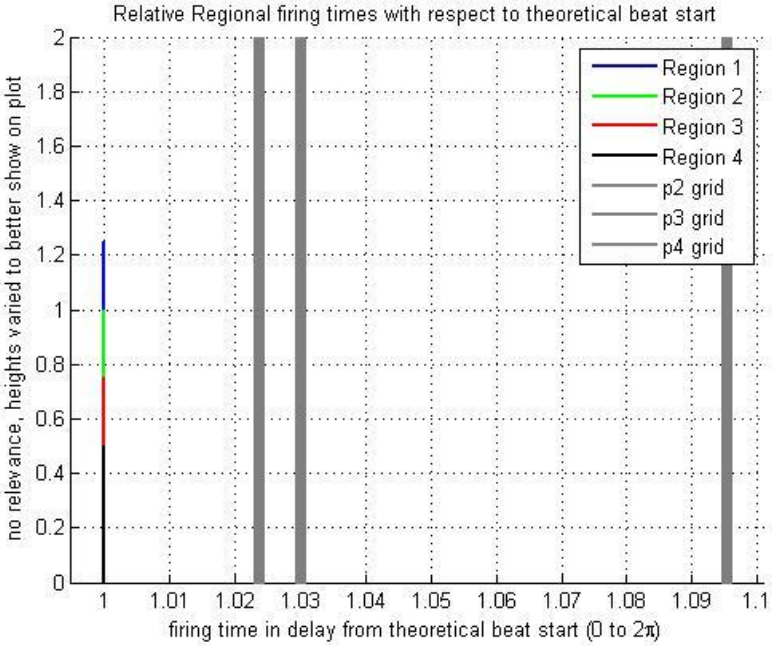

Figure C-26: All Dyssynchrony Regions 2, 3, and 4 (3,1) Timing for No Adjustment

APPENDIX D. GA CODE

D.1. GA Base Code Sets

D.1.1. GA 2 Parameter Automation Code: 3 fixed, 1 variable

```
%Peter Hettwer
%Thesis Work
%Genetic Algorithm Automated Run Sequence
%3 fixed and 1 variable regions

%Prior to execution, user or user generated script file should open a
%parallel MATLAB workstation with the following command:
%matlabpool('open',4); where the number is the number of MATLAB cores
%desired to run.  Maximum Number of cores is dictated by the number of
%physical cores on the computer running MATLAB.  After execution, the user
%or a user generated script file should close the parallel MATLAB
%workstation with the following command: matlabpool('close').

function [] = GA_1_auto_run_A(file, num_runs, noise, seed)
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max ...
   e3max e4max bpm step_size cycle t

run_time = zeros(1,num_runs); %stores run time for all GA run sets
run_gen = zeros(1,num_runs); %stores number of generations for all GA run
%sets
run_cost = zeros(1,num_runs); %stores minimum cost for all GA run sets
p1_hist = zeros(1,num_runs); %region 1 timings for all GA run sets
p2_hist = zeros(1,num_runs); %region 2 timings for all GA run sets
p3_hist = zeros(1,num_runs); %region 3 timings for all GA run sets
p4_hist = zeros(1,num_runs); %region 4 timings for all GA run sets

%Scripted_Initialization; %this CD model parameter initialization file is
%needed if this code is not called from a script file previously calling
%the initialization file.

ti = 0:step_size:1; %set period time values based on desired step size;
%used to create acceleration wave plots.

for k = 1:num_runs; %run the GA code a user specified number of times
   p1_ref = 1;  %region 1 timing is assumed as reference,
   %regions 2, 3, and 4 are controlled within the GA search code.
   %for this variation, regions 3 and 4 are set equal to region 1 internal
   %to the GA search function and region 2 is to be optimized.

   %run the GA search function
```

```
    [run_gen(k) run_cost(k) run_time(k) phase_y(k) amp_y(k) p1_hist(k) ...
        p2_hist(k) p3_hist(k) p4_hist(k)] = GA_1_A(k, noise, ti, ...
        p1_ref,step_size,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,...
        e3min,e3max,e4min,e4max,m1,m2,m3,m4);

    %create acceleration waveforms for the returned values from the GA
    %search function.
    [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1_hist(k),...
        p2_hist(k),p3_hist(k),p4_hist(k));
end

min_run_gen = min(run_gen) %minimum number of run generations from all GA
%runs
max_run_gen = max(run_gen) %maximum number of run generations from all GA
%runs
mean_run_gen = mean(run_gen) % mean number of generations from all GA runs
min_run_cost = min(run_cost) %minimum run cost from all GA runs
max_run_cost = max(run_cost) %maximum run cost from all GA runs
mean_run_cost = mean(run_cost) %mean run cost from all GA runs
min_run_time = min(run_time) %minimum run time from all GA runs
max_run_time = max(run_time)%maximum run time from all GA runs
mean_run_time = mean(run_time) %mean run time from all GA runs
total_run_time = sum(run_time) %sum of all run times from all GA runs

save(file); %save to user defined filename

end
```

D.1.2. GA 2 Parameter Automation Code: 2 fixed, 2 variable

```
%Peter Hettwer
%Thesis Work
%Genetic Algorithm Automated Run Sequence
%2 fixed and 2 variable regions

%Prior to execution, user or user generated script file should open a
%parallel MATLAB workstation with the following command:
%matlabpool('open',4); where the number is the number of MATLAB cores
%desired to run.  Maximum Number of cores is dictated by the number of
%physical cores on the computer running MATLAB.  After execution, the user
%or a user generated script file should close the parallel MATLAB
%workstation with the following command: matlabpool('close').

function [] = GA_1_auto_run_2and2_A(file, num_runs, noise, seed)
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max...
    e3max e4max bpm step cycle t

run_time = zeros(1,num_runs); %stores run time for all GA run sets
run_gen = zeros(1,num_runs); %stores number of generations for all GA run
%sets
run_cost = zeros(1,num_runs); %stores minimum cost for all GA run sets
p1_hist = zeros(1,num_runs); %region 1 timings for all GA run sets
p2_hist = zeros(1,num_runs); %region 2 timings for all GA run sets
p3_hist = zeros(1,num_runs); %region 3 timings for all GA run sets
p4_hist = zeros(1,num_runs); %region 4 timings for all GA run sets

%Scripted_Initialization; %this CD model parameter initialization file is
%needed if this code is not called from a script file previously calling
%the initialization file.

ti = 0:step_size:1; %set period time values based on desired step size;
%used to create acceleration wave plots.

for k = 1:num_runs; %run the GA code a user specified number of times
    p1_ref = 1;  %region 1 timing is assumed as reference,
    %regions 2, 3, and 4 are controlled within the GA search code.
    %for this variation, region 3 is set equal to region 1 and region 4 is
    %set equal to region 2 internal to the GA search function and region 2
    %is to be optimized.

    %run the GA search function
    [run_gen(k) run_cost(k) run_time(k) phase_y(k) amp_y(k) p1_hist(k)...
        p2_hist(k) p3_hist(k) p4_hist(k)] = GA_1_A_2and2(k,noise,ti,p1,...
        step,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,e3min,e3max,...
```

```
        e4min,e4max,m1,m2,m3,m4);

    %create acceleration waveforms for the returned values from the GA
    %search function.
    [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1_hist(k),...
        p2_hist(k),p3_hist(k),p4_hist(k),step,cycle,bpm,r1,r2,r3,r4,...
        e1min,e1max,e2min,e2max,e3min,e3max,e4min,e4max,m1,m2,m3,m4);
end

min_run_gen = min(run_gen) %minimum number of run generations from all GA
%runs
max_run_gen = max(run_gen) %maximum number of run generations from all GA
%runs
mean_run_gen = mean(run_gen) % mean number of generations from all GA runs
min_run_cost = min(run_cost) %minimum run cost from all GA runs
max_run_cost = max(run_cost) %maximum run cost from all GA runs
mean_run_cost = mean(run_cost) %mean run cost from all GA runs
min_run_time = min(run_time) %minimum run time from all GA runs
max_run_time = max(run_time)%maximum run time from all GA runs
mean_run_time = mean(run_time) %mean run time from all GA runs
total_run_time = sum(run_time) %sum of all run times from all GA runs

save(file); %save to user defined filename

end
```

D.1.3. GA 4 Parameter Automation Code

```
%Peter Hettwer
%Thesis Work
%Genetic Algorithm Automated Run Sequence
%Attempt 11

function [] = GA_1_auto_run_4param_A(file, num_runs, noise, seed)
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max...
   e3max e4max bpm step cycle t

run_time = zeros(1,num_runs); %stores run time for all GA run sets
run_gen = zeros(1,num_runs); %stores number of generations for all GA run
%sets
run_cost = zeros(1,num_runs); %stores minimum cost for all GA run sets
p1_hist = zeros(1,num_runs); %region 1 timings for all GA run sets
p2_hist = zeros(1,num_runs); %region 2 timings for all GA run sets
p3_hist = zeros(1,num_runs); %region 3 timings for all GA run sets
p4_hist = zeros(1,num_runs); %region 4 timings for all GA run sets

%Scripted_Initialization; %this CD model parameter initialization file is
%needed if this code is not called from a script file previously calling
%the initialization file.

ti = 0:step_size:1; %set period time values based on desired step size;
%used to create acceleration wave plots.

for k = 1:num_runs; %run the GA code a user specified number of times
   p1_ref = 1;  %region 1 timing is assumed as reference,
   %regions 2, 3, and 4 are controlled within the GA search code.
   %for this variation, all regions 2, 3, and 4 are to be optimized.

   %run the GA search function
   [run_gen(k) run_cost(k) run_time(k) phase_y(k) amp_y(k) p1_hist(k)...
      p2_hist(k) p3_hist(k) p4_hist(k)] = GA_1_A_4param(k,noise,ti,...
      p1_ref,step,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,...
      e3min,e3max,e4min,e4max,m1,m2,m3,m4);

   %create acceleration waveforms for the returned values from the GA
   %search function.
   [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1_hist(k),...
      p2_hist(k),p3_hist(k),p4_hist(k),step,cycle,bpm,r1,r2,r3,r4,...
      e1min,e1max,e2min,e2max,e3min,e3max,e4min,e4max,m1,m2,m3,m4);
end

min_run_gen = min(run_gen) %minimum number of run generations from all GA
```

```
%runs
max_run_gen = max(run_gen) %maximum number of run generations from all GA
%runs
mean_run_gen = mean(run_gen) % mean number of generations from all GA runs
min_run_cost = min(run_cost) %minimum run cost from all GA runs
max_run_cost = max(run_cost) %maximum run cost from all GA runs
mean_run_cost = mean(run_cost) %mean run cost from all GA runs
min_run_time = min(run_time) %minimum run time from all GA runs
max_run_time = max(run_time)%maximum run time from all GA runs
mean_run_time = mean(run_time) %mean run time from all GA runs
total_run_time = sum(run_time) %sum of all run times from all GA runs

save(file); %save to user defined filename

end
```

D.1.4. GA 2 Parameter Base Code: 3 fixed, 1 variable

```
%Peter Hettwer
%Thesis Work
%Genetic Algorithm Backbone Code
%3 fixed and 1 variable region

function [gen_count final_cost time phase amplitude p1_best p2_best ...
    p3_best p4_best] = GA_1_A(seed, noise, t, p1_ref,step,cycle,bpm,r1,...
    r2,r3,r4,e1min,e1max,e2min,e2max,e3min,e3max,e4min,e4max,m1,m2,m3,m4)

tic; %start MATLAB timer to measure how long search takes

%crossover type: single point crossover, binary strings
%parameter encoding scheme: binary
%ranking method: inversely proportional to normalized fit
%pairing method: weighted random based on rank
%fitness function: Least Mean Square
%model: Simple Sinusoid "Acceleration"


%Citation: portions adopted from Jenna Carr, An Introduction to Genetic
%Algorithms; carrjk.pdf Example 2.2

%initialize rng value
%rng(seed); %newer matlab 2013+

SeedRNG(seed); %older matlab: 2010a

%% User Defined Parameters of Genetic Algorithm

%These should be tuned for each model/objective function for best
%performance.

%Max Heart Rate ~250 bpm ~= 4.2  bps = 4.2 Hz  ==>  240 ms period
%Min Heart Rate ~40  bpm ~= 0.67 bps = 0.67 Hz ==> 1500 ms period
%assume want <= 1ms steps; need >= 1500 steps to accomodate min Heart Rate
%>= 11 bits to represent 1500 steps

param_len = 12; %#bits used to represent a time shift;
%12 bits ==> 4096 steps
%4096 steps ==> minimum 0.366211 ms/step @ 40 bpm

population_size = 20; %Genetic Algorithm population size

elitism = 0.1; %Top n of previous generation passed on to next gen
```

```
%unchanged ex. 0.1 ==> 10%

Pcross = 1; % % of new generation made up of offspring
%of previous generation ex. 1 - 0.1 = 0.9 ==> 90%

Pmute = 0.001; %chance of flipping a bit, 0.1%

n_param = 2; %number of parameters in each chromosome/test case

Nbits_total = n_param * param_len; %# bits in a test case
%= # bits per parameter * # parameters

gen_max = 50; %maximum number of generations

min_cost = 0; %set min cost value for early termination of GA.

noise_max = noise;

% Parameters based on User Defined Parameters

converted_parameter_list = zeros(population_size,n_param); %decimal
%representation of chromosomes
elite = ceil(elitism * population_size); % # of elite saved each generation
offspring = population_size - elite; %# of offspring to generate on every
%mating mix
gen_count = 0; %generation counter instantiated and initialized to 0
stop_count = 0; %counter to terminate GA run if best solution doesn't
%improve for stop_gen generations
stop_gen = 5; %GA terminates if no solution improvement seen for this
%number of generations.
population = InitPopulation(population_size, Nbits_total); %generate the
%initial, random population

p1_bin = fix2bin(p1_ref,0,0.9998*2*pi,param_len); %create the binary value
%of the supplied reference region timing of the set binary length

%% Parameters to be optimized
A1 = zeros(population_size, length(t)); %Acceleration 1 for cardiac
%dyssynchrony model or sin 1 for simple model
A2 = zeros(population_size, length(t)); %Acceleration 2 for cardiac
%dyssynchrony model or sin 2 for simple model
A3 = zeros(population_size, length(t)); %Acceleration 3 for cardiac
%dyssynchrony model
A4 = zeros(population_size, length(t)); %Acceleration 4 for cardiac
%dyssynchrony model
```

```
%% Parameter Envelopes

phase_max = 2*pi; %param 1 max
phase_min = 0; %param 1 min

%% Initial objective function evaluations

%Transform test cases to usable values and run through model function
for k = 1:population_size
   population(k,1:param_len) = p1_bin; %copy reference phase into each
   %chromosome

   converted_parameter_list(k,:) = bin2fix(population(k,:),phase_min,...
      phase_max,n_param,phase_min,phase_max);

   p1(k) = converted_parameter_list(k,1); %record region 1 timing values
   p2(k) = converted_parameter_list(k,2); %record region 2 timing values
   p3(k) = p1(k); %p3 reset same as p1 after descritization since not
   %being optimized by GA,
   p4(k) = p1(k); %p4 reset same as p1 after descritization since not
   %being optimized by GA,

end

parfor k = 1:population_size
   [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1(k),p2(k),...
      p3(k),p4(k),step,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,...
      e3min,e3max,e4min,e4max,m1,m2,m3,m4); %find acceleration waveforms

end

for k = 1:population_size
   cost(k) = SSE(A1(k,:), A2(k,:)); %generate costs for each chromosome
end

%(want min cost to have highest prob)
cost_norm = max(cost) - cost; %invert high and low costs for mating
%probability. highest cost is eliminated from propagation
[cost_norm, cost_norm_ind] = sort(cost_norm,'descend'); %search population
%w/ "highest" cost in first entry (lowest actual cost).
population = population(cost_norm_ind,:); %sort population w/ lowest cost
%first
max_obj(gen_count + 1) = max(cost); %contains min of initial population
min_obj(1) = min(cost); %contains min of initial population
mean_obj(1) = mean(cost_norm); %contains mean of initial population
mate_prob = cost_norm/sum(cost_norm); %probability normalization;
```

```
%% Main Loop

while (gen_count < gen_max)

   gen_count = gen_count +1 %increment generation/loop counter

   %Choose Mates:

   M = offspring / 2; %number of parings

   [mate1, mate2] = Mates(mate_prob,M); %generate pairing vectors

   cross_point = Cross(M,Pcross,Nbits_total); %generate pairing crossover
   %points

   %generate next generation's population
   population = NextGen(population,elite,cross_point,mate1,mate2);

   %protect elite population from mutation:

   for k = 1:elite
      pop_temp(k) = population(k);
   end

   %Mutate Population:

   mutation = Mutate(population,Pmute);

   population = abs(population - mutation);
     %toggle bit at specified location

   %rewrite elite population
   for k = 1:elite
      population(k) = pop_temp(k);
   end


   %% Evaluate New Population for cost:

   %Transform test cases to usable values and run through model function
for k = 1:population_size
   population(k,1:param_len) = p1_bin; %copy reference phase into each
   %chromosome

   converted_parameter_list(k,:) = bin2fix(population(k,:),phase_min,...
```

```
        phase_max,n_param,phase_min,phase_max);

    p1(k) = converted_parameter_list(k,1); %record region 1 timing values
    p2(k) = converted_parameter_list(k,2); %record region 2 timing values
    p3(k) = p1(k); %p3 reset same as p1 after descritization since not
    %being optimized by GA,
    p4(k) = p1(k); %p4 reset same as p1 after descritization since not
    %being optimized by GA,

end

parfor k = 1:population_size
    [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1(k),p2(k),...
        p3(k),p4(k),step,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,...
        e3min,e3max,e4min,e4max,m1,m2,m3,m4); %find acceleration waveforms

end

for k = 1:population_size
    cost(k) = SSE(A1(k,:), A2(k,:)); %generate costs for each chromosome
end

%(want min cost to have highest prob)
cost_norm = max(cost) - cost; %invert high and low costs for mating
%probability. highest cost is eliminated from propagation
[cost_norm, cost_norm_ind] = sort(cost_norm,'descend'); %search population
%w/ "highest" cost in first entry (lowest actual cost).
population = population(cost_norm_ind,:); %sort population w/ lowest cost
%first
max_obj(gen_count + 1) = max(cost); %contains min of current population
min_obj(gen_count + 1) = min(cost); %contains min of current population
mean_obj(gen_count + 1) = mean(cost_norm); %contains mean of current
%population normalized
mate_prob = cost_norm/sum(cost_norm); %probability normalization;

%% Early Stopping Criteria:

%increment stop counter if improvement not made in current generation
if( min_obj(gen_count) <= min_obj(gen_count+1) )
    stop_count = stop_count + 1;
else
    stop_count = 0;
end


if ( (gen_count> gen_max) || min_obj(gen_count+1) <= 0.01 || ...
```

```
      stop_count == stop_gen )

   break
end

end %while
time = toc; %stop MATLAB Timer and display to command window
final_cost = cost(1); %record min cost from GA run
p1_best = p1(1); %record best timing value for region 1
p2_best = p2(1); %record best timing value for region 2
p3_best = p3(1); %record best timing value for region 3
p4_best = p4(1); %record best timing value for region 4

end
```

D.1.5. GA 2 Parameter Base Code: 2 fixed, 2 variable

```
%Peter Hettwer
%Thesis Work
%Genetic Algorithm Backbone Code
%2 fixed and 2 variable regions

function [gen_count final_cost time phase amplitude p1_best p2_best ...
    p3_best p4_best] = GA_1_A_2and2(seed, noise, t, p1_ref,step,cycle,...
    bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,e3min,e3max,e4min,e4max,m1,...
    m2,m3,m4)

tic; %start MATLAB timer to measure how long search takes

%crossover type: single point crossover, binary strings
%parameter encoding scheme: binary
%ranking method: inversely proportional to normalized fit
%pairing method: weighted random based on rank
%fitness function: Least Mean Square
%model: Simple Sinusoid "Acceleration"


%Citation: portions adopted from Jenna Carr, An Introduction to Genetic
%Algorithms; carrjk.pdf Example 2.2

%initialize rng value
%rng(seed); %newer matlab 2013+

SeedRNG(seed); %older matlab: 2010a

%% User Defined Parameters of Genetic Algorithm

%These should be tuned for each model/objective function for best
%performance.

%Max Heart Rate ~250 bpm ~= 4.2  bps = 4.2 Hz  ==>  240 ms period
%Min Heart Rate ~40  bpm ~= 0.67 bps = 0.67 Hz ==> 1500 ms period
%assume want <= 1ms steps; need >= 1500 steps to accomodate min Heart Rate
%>= 11 bits to represent 1500 steps

param_len = 12; %#bits used to represent a time shift;
%12 bits ==> 4096 steps
%4096 steps ==> minimum 0.366211 ms/step @ 40 bpm

population_size = 20; %Genetic Algorithm population size
```

```
elitism = 0.1; %Top n of previous generation passed on to next gen
%unchanged ex. 0.1 ==> 10%

Pcross = 1; % % of new generation made up of offspring
%of previous generation ex. 1 - 0.1 = 0.9 ==> 90%

Pmute = 0.001; %chance of flipping a bit, 0.1%

n_param = 2; %number of parameters in each chromosome/test case

Nbits_total = n_param * param_len; %# bits in a test case
%= # bits per parameter * # parameters

gen_max = 50; %maximum number of generations

min_cost = 0; %set min cost value for early termination of GA.

noise_max = noise;

% Parameters based on User Defined Parameters

converted_parameter_list = zeros(population_size,n_param); %decimal
%representation of chromosomes
elite = ceil(elitism * population_size); % # of elite saved each generation
offspring = population_size - elite; %# of offspring to generate on every
%mating mix
gen_count = 0; %generation counter instantiated and initialized to 0
stop_count = 0; %counter to terminate GA run if best solution doesn't
%improve for stop_gen generations
stop_gen = 5; %GA terminates if no solution improvement seen for this
%number of generations.
population = InitPopulation(population_size, Nbits_total); %generate the
%initial, random population

p1_bin = fix2bin(p1_ref,0,0.9998*2*pi,param_len); %create the binary value
%of the supplied reference region timing of the set binary length

%% Parameters to be optimized
A1 = zeros(population_size, length(t)); %Acceleration 1 for cardiac
%dyssynchrony model or sin 1 for simple model
A2 = zeros(population_size, length(t)); %Acceleration 2 for cardiac
%dyssynchrony model or sin 2 for simple model
A3 = zeros(population_size, length(t)); %Acceleration 3 for cardiac
%dyssynchrony model
A4 = zeros(population_size, length(t)); %Acceleration 4 for cardiac
%dyssynchrony model
```

```
%% Parameter Envelopes

phase_max = 2*pi; %param 1 max
phase_min = 0; %param 1 min

%% Initial objective function evaluations

%Transform test cases to usable values and run through model function
for k = 1:population_size
   population(k,1:param_len) = p1_bin; %copy reference phase into each
   %chromosome

   converted_parameter_list(k,:) = bin2fix(population(k,:),phase_min,...
      phase_max,n_param,phase_min,phase_max);

   p1(k) = converted_parameter_list(k,1) %record region 1 timing values
   p2(k) = converted_parameter_list(k,2) %record region 2 timing values
   p3(k) = p1(k) %p3 reset same as p1 after descritization since not
   %being optimized by GA,
   p4(k) = p2(k) %p4 reset same as p2 after descritization since not
   %being optimized by GA,

end

parfor k = 1:population_size
   [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1(k),p2(k),...
      p3(k),p4(k),step,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,...
      e3min,e3max,e4min,e4max,m1,m2,m3,m4); %find acceleration waveforms

end

for k = 1:population_size
   cost(k) = SSE(A1(k,:), A2(k,:)); %generate costs for each chromosome
end

%(want min cost to have highest prob)
cost_norm = max(cost) - cost; %invert high and low costs for mating
%probability. highest cost is eliminated from propagation
[cost_norm, cost_norm_ind] = sort(cost_norm,'descend'); %search population
%w/ "highest" cost in first entry (lowest actual cost).
population = population(cost_norm_ind,:); %sort population w/ lowest cost
%first
max_obj(gen_count + 1) = max(cost); %contains min of initial population
min_obj(1) = min(cost); %contains min of initial population
mean_obj(1) = mean(cost_norm); %contains mean of initial population
```

```
mate_prob = cost_norm/sum(cost_norm); %probability normalization;
%want min cost to have highest prob.


%% Main Loop

while (gen_count < gen_max)

   gen_count = gen_count +1 %increment generation/loop counter

   %Choose Mates:

   M = offspring / 2; %number of parings

   [mate1, mate2] = Mates(mate_prob,M); %generate pairing vectors

   cross_point = Cross(M,Pcross,Nbits_total); %generate pairing crossover
   %points

   %generate next generation's population
   population = NextGen(population,elite,cross_point,mate1,mate2);

   %protect elite population from mutation:

   for k = 1:elite
      pop_temp(k) = population(k);
   end

   %Mutate Population:

   mutation = Mutate(population,Pmute);

   population = abs(population - mutation);
     %toggle bit at specified location

   %rewrite elite population
   for k = 1:elite
      population(k) = pop_temp(k);
   end


   %% Evaluate New Population for cost:

   %Transform test cases to usable values and run through model function
for k = 1:population_size
   population(k,1:param_len) = p1_bin; %copy reference phase into each
```

```
    %chromosome

    converted_parameter_list(k,:) = bin2fix(population(k,:),phase_min,...
        phase_max,n_param,phase_min,phase_max);

    p1(k) = converted_parameter_list(k,1) %record region 1 timing values
    p2(k) = converted_parameter_list(k,2) %record region 2 timing values
    p3(k) = p1(k) %p3 reset same as p1 after descritization since not
    %being optimized by GA,
    p4(k) = p2(k) %p4 reset same as p2 after descritization since not
    %being optimized by GA,

end

parfor k = 1:population_size
    [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1(k),p2(k),...
        p3(k),p4(k),step,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,...
        e3min,e3max,e4min,e4max,m1,m2,m3,m4); %find acceleration waveforms

end

for k = 1:population_size
    cost(k) = SSE(A1(k,:), A2(k,:)); %generate costs for each chromosome
end

%(want min cost to have highest prob)
cost_norm = max(cost) - cost; %invert high and low costs for mating
%probability. highest cost is eliminated from propagation
[cost_norm, cost_norm_ind] = sort(cost_norm,'descend'); %search population
%w/ "highest" cost in first entry (lowest actual cost).
population = population(cost_norm_ind,:); %sort population w/ lowest cost
%first
max_obj(gen_count + 1) = max(cost); %contains min of current population
min_obj(gen_count + 1) = min(cost); %contains min of current population
mean_obj(gen_count + 1) = mean(cost_norm); %contains mean of current
%population normalized
mate_prob = cost_norm/sum(cost_norm); %probability normalization;

%% Early Stopping Criteria:

%increment stop counter if improvement not made in current generation
if( min_obj(gen_count) <= min_obj(gen_count+1) )
    stop_count = stop_count + 1;
else
    stop_count = 0;
end
```

```
if ( (gen_count> gen_max) || min_obj(gen_count+1) <= 0.01 || ...
      stop_count == stop_gen )

   break
end

end %while
time = toc; %stop MATLAB Timer and display to command window
final_cost = cost(1); %record min cost from GA run
p1_best = p1(1); %record best timing value for region 1
p2_best = p2(1); %record best timing value for region 2
p3_best = p3(1); %record best timing value for region 3
p4_best = p4(1); %record best timing value for region 4

end
```

D.1.6. GA 4 Parameter Base Code

```
%Peter Hettwer
%Thesis Work
%Genetic Algorithm Backbone Code
%1 fixed and 3 variable regions

function [gen_count final_cost time phase amplitude p1_best p2_best ...
    p3_best p4_best] = GA_1_A(seed,noise,t,p1_ref,step,cycle,bpm,r1,r2,...
    r3,r4,e1min,e1max,e2min,e2max,e3min,e3max,e4min,e4max,m1,m2,m3,m4)

tic; %start MATLAB timer to measure how long search takes

%crossover type: single point crossover, binary strings
%parameter encoding scheme: binary
%ranking method: inversely proportional to normalized fit
%pairing method: weighted random based on rank
%fitness function: Least Mean Square
%model: Simple Sinusoid "Acceleration"


%Citation: portions adopted from Jenna Carr, An Introduction to Genetic
%Algorithms; carrjk.pdf Example 2.2

%initialize rng value
%rng(seed); %newer matlab 2013+

SeedRNG(seed); %older matlab: 2010a

%% User Defined Parameters of Genetic Algorithm

%These should be tuned for each model/objective function for best
%performance.

%Max Heart Rate ~250 bpm ~= 4.2  bps = 4.2 Hz  ==>  240 ms period
%Min Heart Rate ~40  bpm ~= 0.67 bps = 0.67 Hz ==> 1500 ms period
%assume want <= 1ms steps; need >= 1500 steps to accomodate min Heart Rate
%>= 11 bits to represent 1500 steps

param_len = 12; % #bits used to represent a time shift;
%12 bits ==> 4096 steps
%4096 steps ==> minimum 0.366211 ms/step @ 40 bpm

population_size = 20; %Genetic Algorithm population size

elitism = 0.1; %Top n of previous generation passed on to next gen
```

```
%unchanged ex. 0.1 ==> 10%

Pcross = 1; % % of new generation made up of offspring
%of previous generation ex. 1 - 0.1 = 0.9 ==> 90%

Pmute = 0.001; %chance of flipping a bit, 0.1%

n_param = 4; %number of parameters in each chromosome/test case

Nbits_total = n_param * param_len; %# bits in a test case
%= # bits per parameter * # parameters

gen_max = 50; %maximum number of generations

min_cost = 0; %set min cost value for early termination of GA.

noise_max = noise;

% Parameters based on User Defined Parameters

converted_parameter_list = zeros(population_size,n_param); %decimal
%representation of chromosomes
elite = ceil(elitism * population_size); % # of elite saved each generation
offspring = population_size - elite; %# of offspring to generate on every
%mating mix
gen_count = 0; %generation counter instantiated and initialized to 0
stop_count = 0; %counter to terminate GA run if best solution doesn't
%improve for stop_gen generations
stop_gen = 5; %GA terminates if no solution improvement seen for this
%number of generations.
population = InitPopulation(population_size, Nbits_total); %generate the
%initial, random population

p1_bin = fix2bin(p1_ref,0,0.9998*2*pi,param_len); %create the binary value
%of the supplied reference region timing of the set binary length

%% Parameters to be optimized
A1 = zeros(population_size, length(t)); %Acceleration 1 for cardiac
%dyssynchrony model or sin 1 for simple model
A2 = zeros(population_size, length(t)); %Acceleration 2 for cardiac
%dyssynchrony model or sin 2 for simple model
A3 = zeros(population_size, length(t)); %Acceleration 3 for cardiac
%dyssynchrony model
A4 = zeros(population_size, length(t)); %Acceleration 4 for cardiac
%dyssynchrony model
```

```
%% Parameter Envelopes

phase_max = 2*pi; %param 1 max
phase_min = 0; %param 1 min

%% Initial objective function evaluations

%Transform test cases to usable values and run through model function
for k = 1:population_size
   population(k,1:param_len) = p1_bin; %copy reference phase into each
   %chromosome

   converted_parameter_list(k,:) = bin2fix_4param(population(k,:),...
      phase_min,phase_max,n_param,phase_min,phase_max);

   p1(k) = converted_parameter_list(k,1) %record region 1 timing values
   p2(k) = converted_parameter_list(k,2) %record region 2 timing values
   p3(k) = converted_parameter_list(k,3) %record region 3 timing values
   p4(k) = converted_parameter_list(k,4) %record region 4 timing values
end

parfor k = 1:population_size
   [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1(k),p2(k),...
      p3(k),p4(k),step,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,...
      e3min,e3max,e4min,e4max,m1,m2,m3,m4); %find acceleration waveforms

end

for k = 1:population_size
   cost(k) = SSE(A1(k,:), A2(k,:)); %generate costs for each chromosome
end

%(want min cost to have highest prob)
cost_norm = max(cost) - cost; %invert high and low costs for mating
%probability. highest cost is eliminated from propagation

[cost_norm, cost_norm_ind] = sort(cost_norm,'descend'); %search population
%w/ "highest" cost in first entry (lowest actual cost).

population = population(cost_norm_ind,:); %sort population w/ lowest cost
%first

max_obj(gen_count + 1) = max(cost); %contains min of initial population
min_obj(1) = min(cost); %contains min of initial population
mean_obj(1) = mean(cost_norm); %contains mean of initial population
mate_prob = cost_norm/sum(cost_norm); %probability normalization
```

```
%% Main Loop

while (gen_count < gen_max)

   gen_count = gen_count +1 %increment generation/loop counter

   %Choose Mates:

   M = offspring / 2; %number of parings

   [mate1, mate2] = Mates(mate_prob,M); %generate pairing vectors

   cross_point = Cross(M,Pcross,Nbits_total); %generate pairing crossover
   %points

   %generate next generation's population
   population = NextGen(population,elite,cross_point,mate1,mate2);

   %protect elite population from mutation:

   for k = 1:elite
      pop_temp(k) = population(k);
   end

   %Mutate Population:

   mutation = Mutate(population,Pmute);

   population = abs(population - mutation);
     %toggle bit at specified location

   %rewrite elite population
   for k = 1:elite
      population(k) = pop_temp(k);
   end


   %% Evaluate New Population for cost:

   %Transform test cases to usable values and run through model function
for k = 1:population_size
   population(k,1:param_len) = p1_bin; %copy reference phase into each
   %chromosome

   converted_parameter_list(k,:) = bin2fix(population(k,:),phase_min,...
```

```
        phase_max,n_param,phase_min,phase_max);

    p1(k) = converted_parameter_list(k,1) %record region 1 timing values
    p2(k) = converted_parameter_list(k,2) %record region 2 timing values
    p3(k) = converted_parameter_list(k,3) %record region 3 timing values
    p4(k) = converted_parameter_list(k,4) %record region 4 timing values
end

parfor k = 1:population_size
    [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1(k),p2(k),...
        p3(k),p4(k),step,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,...
        e3min,e3max,e4min,e4max,m1,m2,m3,m4); %find acceleration waveforms

end

for k = 1:population_size
    cost(k) = SSE(A1(k,:), A2(k,:)); %generate costs for each chromosome
end

%(want min cost to have highest prob)
cost_norm = max(cost) - cost; %invert high and low costs for mating
%probability. highest cost is eliminated from propagation
[cost_norm, cost_norm_ind] = sort(cost_norm,'descend'); %search population
%w/ "highest" cost in first entry (lowest actual cost).
population = population(cost_norm_ind,:); %sort population w/ lowest cost
%first
max_obj(gen_count + 1) = max(cost); %contains min of current population
min_obj(gen_count + 1) = min(cost); %contains min of current population
mean_obj(gen_count + 1) = mean(cost_norm); %contains mean of current
%population normalized
mate_prob = cost_norm/sum(cost_norm); %probability normalization;

%% Early Stopping Criteria:

%increment stop counter if improvement not made in current generation
if( min_obj(gen_count) <= min_obj(gen_count+1) )
    stop_count = stop_count + 1;
else
    stop_count = 0;
end


if ( (gen_count> gen_max) || min_obj(gen_count+1) <= 0.01 || ...
        stop_count == stop_gen )

    break
```

```
end

end %while
time = toc; %stop MATLAB Timer and display to command window
final_cost = cost(1); %record min cost from GA run
p1_best = p1(1); %record best timing value for region 1
p2_best = p2(1); %record best timing value for region 2
p3_best = p3(1); %record best timing value for region 3
p4_best = p4(1); %record best timing value for region 4

end
```

D.2. GA Function Calls

D.2.1. bin2fix.m

```
%Peter Hettwer
%Thesis Work
%2 Parameter Binary to Unsigned Fixed Point Conversion

function Value = bin2fix(Bin,Min,Max,n_param,amp_min,amp_max)

%Take a binary number, Bin, and convert it to one value in the range of Min
% to Max and a second value in the range of amp_min to amp_max

[pop_size bits] = size(Bin); %obtain population size and number of bits
%in each chromosome from the binary matrix passed to the function.

param_length = bits/n_param; %obtain the bit lenght of each parameter
%from number of parameters contained within each chromosome.

temp_sum = zeros(pop_size,n_param); %contains the matrix of temporary sums
%to compute fixed point values for each parameter in each chromosome.

for n = 1:n_param %step through chromosome parameters
   for k = 1:param_length %step through individual parameter bits
      %compute sums for each parameter in each chromosome of the
      %population in the range of 0 to (1-2^(-bits)).
      temp_sum(:,n) = temp_sum(:,n) + ...
         2^(-(k))*Bin(:,((n-1)*param_length + k));
   end

end

%Take normalized sum, scales to the desired range, and returns to the
%calling function.
Value(:,1) = Min + (Max - Min)*temp_sum(:,1); %value for p1
Value(:,2) = amp_min + (amp_max - amp_min)*temp_sum(:,2); %value for p2

end
```

D.2.2. Bin2fix_4param.m

```
%Peter Hettwer
%Thesis Work
%Genetic Algorithm 4 Parameter Binary to Unsigned Fixed Point Conversion

function Value = bin2fix_4param(Bin,Min,Max,n_param,amp_min,amp_max)

%Take a binary number, Bin, and convert it to a value in the range of Min
% to Max.  Code allows for varying the range of region 2 (region used for
% dyssynchrony test cases), but this functionality not used.

[pop_size bits] = size(Bin); %obtain population size and number of bits
%in each chromosome from the binary matrix passed to the function.

param_length = bits/n_param; %obtain the bit lenght of each parameter
%from number of parameters contained within each chromosome.

temp_sum = zeros(pop_size,n_param);%contains the matrix of temporary sums
%to compute fixed point values for each parameter in each chromosome.

for n = 1:n_param %step through chromosome parameters
   for k = 1:param_length %step through individual parameter bits
      %compute sums for each parameter in each chromosome of the
      %population in the range of 0 to (1-2^(-bits)).
      temp_sum(:,n) = temp_sum(:,n) + ...
         2^(-(k))*Bin(:,((n-1)*param_length + k));
   end

end

Value(:,1) = Min + (Max - Min)*temp_sum(:,1); %value for p1
Value(:,2) = amp_min + (amp_max - amp_min)*temp_sum(:,2); %value for p2
Value(:,3) = Min + (Max - Min)*temp_sum(:,3); %value for p3
Value(:,4) = Min + (Max - Min)*temp_sum(:,4); %value for p4
end
```

D.2.3. Cross.m

```
%Peter Hettwer
%Thesis Work
%Crossover Selection Function



function crossOverPoint = Cross(nPairs, Pcross, bitsTotal)
   %generate random crossover points for each pair of mating chromosomes
   crossOverPoint = ceil(rand(1,nPairs)*(bitsTotal - 1))...
      .*(rand(1,nPairs) < Pcross);
   %rounds up random number to be between 1 and Nbits_total. value used
   %to select crossover index.  All crossover points selected at once.


   for k = 1:nPairs
      %fail safe check to ensure no crossover point is zero (caused if
      %rand returns 0). If such a case does occur (very rare), places
      %that crossover point at the final bit position to allow proper
      %indexing in calling code.
      if(crossOverPoint(k) ==0)
         crossOverPoint(k) = bitsTotal;
      end
   end
end
```

D.2.4. fix2bin.m

```
%Peter Hettwer
%Thesis Work
%Generic Unsigned Fixed Point to Binary Conversion

function Value = fix2bin(Fix,Min,Max,bits)

%Take a number, Fix, in the range of Min to Max and convert it to a binary
%value with the supplied number of bits

temp = zeros(1,bits); %temporary vector to store binary bit values
norm = (Fix-Min)./(Max - Min); %normalized version of number supplied for
%conversion


   for k = 1:bits %determine individual bit values starting with most
      %significant bit
      if( (norm - 2^(-(k))) > 0)
         temp(k) = 1;
         norm = norm - 2^(-(k));
      else
         temp(k) = 0;
      end
   end

%Return the converted number as a binary string
Value = temp;

end
```

D.2.5. initPopulation.m

```
%Peter Hettwer
%Thesis Work
%Initial Binary Population Creation
%Attempt 0

function initial_population = InitPopulation(populationSize, numBits)
   %creates a random, binary population of the desired size and number of
   %bits.

   %generates random values of 0 or 1 for each position in a matrix of
   %size populationSize by numBits.
   initial_population = round(rand(populationSize,numBits));
end
```

D.2.6. Mates.m

```
%Peter Hettwer
%Thesis Work
%Mate Selection

%generates two vectors, mate1 and mate 2 for GA chromosome crossover
%pairing selection based on supplied mating selection probability of each
%chromosome.  Code allows for chromosome pairing with iteself.

%uses RandChooseN.m function developed by Jenna Carr.  Available at:
%Citation:
%http://people.whitman.edu/~hundledr/courses/M350/RandChooseN.m

function [mate1, mate2] = Mates(mating_prob, numMates)
   mate1 = RandChooseN(mating_prob,numMates); %generate mating vector 1
   mate2 = RandChooseN(mating_prob,numMates); %generate mating vector 2
end
```

D.2.7. NextGen.m

```
%Peter Hettwer
%Thesis Work
%Next Generation Creation

function nextGeneration = NextGen(currentGen,numElite,crossPoint,mate1,mate2)
   %start at index values after the elite population to be saved from one
   %population to the next.
   genSize = size(currentGen); %determine size of current generation

   %obtain population size from supplied current generation
   popSize = genSize(1);

   %obtain number of bits in each chromosome from supplied current
   %generation
   numBits = genSize(2);

   %Rewrite current generation to next generation to save elite population
   nextGeneration = currentGen;

   %perform crossover of odd numbered current generation mating pairs at
   %specified crossover point after elite population
   nextGeneration((1+numElite):2:popSize,:) = ...
      [currentGen(mate1,1:crossPoint()) ...
      currentGen(mate2,crossPoint()+1:numBits)];

   %perform crossover of odd numbered current generation mating pairs at
   %specified crossover point after elite population
   nextGeneration((2+numElite):2:popSize,:) = ...
      [currentGen(mate2,1:crossPoint()) ...
      currentGen(mate1,crossPoint()+1:numBits)];
end
```

D.2.8. SeedRNG.m

```
%Peter Hettwer
%Thesis Work
%MATLAB RNG seeding function

%seeds MATLAB RNG to allow for reproducible results in any function
%requiring use of rand and randn functions.

function [] = SeedRNG(integer)
   stream = RandStream('mt19937ar','Seed',integer);
   RandStream.setDefaultStream(stream);
end
```

D.2.9. SSE.m

```
%Peter Hettwer
%Thesis Work
%Sum Squared Error Optimization Function for 2 parameter optimization

function fit = SSE(ideal, experimental)
%ideal is the reference value, experimental is the value being optimized
diff = ideal - experimental;
fit = sum(diff.^2);
end
```

D.2.10. SSE4.m

```
%Peter Hettwer
%Thesis Work
%Sum Squared Error Optimization Function for 4 parameter optimization

%ideal is the reference value (region 1), experiemntal1 is region 2,
%experimental2 is region 3, and experimental3 is region4

function fit = SSE(ideal, experimental1, experimental2, experimental3)

diff1 = ideal - experimental1;
diff2 = ideal - experimental2;
diff3 = ideal - experimental3;
fit = sum((diff1.^2 + diff2.^2 + diff3.^2));
end
```

D.2.11. Scripted_Initialization_no_dyss.m

```
% Scripted Initialization Functions
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max e3max e4max bpm
step cycle t

%The Resistance Values (which ranges from 0.00001 - 0.0075)
r1 = 0.0015;  r2 = 0.0015;  r3 = 0.0015;  r4 = 0.0015;
%r= 0.00001 + (0.0075-0.00001).*rand; % random generator for the
%Resistance
%r1= r;  r2= r;  r3= r;  r4=r;

%The Mass Values (which ranges from 0.0001 - 0.01)
m1 = 0.001; m2 = 0.001; m3 = 0.001; m4 = 0.001;
%m= 0.0001 + (0.01-0.0001).*rand;  % random generator for the masses
%m1= m;  m2= m;  m3= m;  m4=m;

%The Min Elastance Values (which ranges from 1 - 5)
e1min = 1;    e2min = 1;    e3min = 1;    e4min = 1;
%emin= 1 + (5-1).*rand; % random generator for the
%Min Elastance
%emin1= emin;  emin2= emin;  emin3= emin;  emin4=emin;

%The Max Elastance Values (which ranges from 10 - 50)
e1max = 20; e2max = 20; e3max = 20; e4max = 20;
%emax= 10 + (50-10).*rand; % random generator for the
%Max Elastance
%emax1= emax;  emax2= emax;  emax3= emax;  emax4=emax;

%Heart Perimeters
bpm = 60;   step = 0.001;   cycle = 5;
```

D.2.12. Scripted_Initialization_r2_dyss_0_015.m

```
% Scripted Initialization Functions
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max e3max e4max bpm
step cycle t

%The Resistance Values (which ranges from 0.00001 - 0.0075)
r1 = 0.0015;  r2 = 0.015;  r3 = 0.0015;  r4 = 0.0015;
%r= 0.00001 + (0.0075-0.00001).*rand; % random generator for the
%Resistance
%r1= r;  r2= r;  r3= r;  r4=r;

%The Mass Values (which ranges from 0.0001 - 0.01)
m1 = 0.001; m2 = 0.001; m3 = 0.001; m4 = 0.001;
%m= 0.0001 + (0.01-0.0001).*rand;  % random generator for the masses
%m1= m;  m2= m;  m3= m;  m4=m;

%The Min Elastance Values (which ranges from 1 - 5)
e1min = 1;    e2min = 1;    e3min = 1;    e4min = 1;
%emin= 1 + (5-1).*rand; % random generator for the
%Min Elastance
%emin1= emin;  emin2= emin;  emin3= emin;  emin4=emin;

%The Max Elastance Values (which ranges from 10 - 50)
e1max = 20; e2max = 20; e3max = 20; e4max = 20;
%emax= 10 + (50-10).*rand; % random generator for the
%Max Elastance
%emax1= emax;  emax2= emax;  emax3= emax;  emax4=emax;

%Heart Perimeters
bpm = 60;   step = 0.001;   cycle = 5;
```

D.2.13. Scripted_Initialization_m2_dyss_0_01.m

```
% Scripted Initialization Functions
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max e3max e4max bpm
step cycle t

%The Resistance Values (which ranges from 0.00001 - 0.0075)
r1 = 0.0015;  r2 = 0.0015;  r3 = 0.0015;  r4 = 0.0015;
%r= 0.00001 + (0.0075-0.00001).*rand; % random generator for the
%Resistance
%r1= r;  r2= r;  r3= r;  r4=r;

%The Mass Values (which ranges from 0.0001 - 0.01)
m1 = 0.001; m2 = 0.01; m3 = 0.001; m4 = 0.001;
%m= 0.0001 + (0.01-0.0001).*rand;  % random generator for the masses
%m1= m;  m2= m;  m3= m;  m4=m;

%The Min Elastance Values (which ranges from 1 - 5)
e1min = 1;    e2min = 1;    e3min = 1;    e4min = 1;
%emin= 1 + (5-1).*rand; % random generator for the
%Min Elastance
%emin1= emin;  emin2= emin;  emin3= emin;  emin4=emin;

%The Max Elastance Values (which ranges from 10 - 50)
e1max = 20; e2max = 20; e3max = 20; e4max = 20;
%emax= 10 + (50-10).*rand; % random generator for the
%Max Elastance
%emax1= emax;  emax2= emax;  emax3= emax;  emax4=emax;

%Heart Perimeters
bpm = 60;   step = 0.001;   cycle = 5;
```

D.2.14. Scripted_Initialization_min_elas2_dyss_4.m

```
% Scripted Initialization Functions
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max e3max e4max bpm
step cycle t

%The Resistance Values (which ranges from 0.00001 - 0.0075)
r1 = 0.0015;  r2 = 0.0015;  r3 = 0.0015;  r4 = 0.0015;
%r= 0.00001 + (0.0075-0.00001).*rand; % random generator for the
%Resistance
%r1= r;  r2= r;  r3= r;  r4=r;

%The Mass Values (which ranges from 0.0001 - 0.01)
m1 = 0.001; m2 = 0.001; m3 = 0.001; m4 = 0.001;
%m= 0.0001 + (0.01-0.0001).*rand;  % random generator for the masses
%m1= m;  m2= m;  m3= m;  m4=m;

%The Min Elastance Values (which ranges from 1 - 5)
e1min = 1;    e2min = 4;    e3min = 1;    e4min = 1;
%emin= 1 + (5-1).*rand; % random generator for the
%Min Elastance
%emin1= emin;  emin2= emin;  emin3= emin;  emin4=emin;

%The Max Elastance Values (which ranges from 10 - 50)
e1max = 20; e2max = 20; e3max = 20; e4max = 20;
%emax= 10 + (50-10).*rand; % random generator for the
%Max Elastance
%emax1= emax;  emax2= emax;  emax3= emax;  emax4=emax;

%Heart Perimeters
bpm = 60;   step = 0.001;   cycle = 5;
```

D.2.15. Scripted_Initialization_max_elas2_dyss_40.m

```
% Scripted Initialization Functions
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max e3max e4max bpm
step cycle t

%The Resistance Values (which ranges from 0.00001 - 0.0075)
r1 = 0.0015;  r2 = 0.0015;  r3 = 0.0015;  r4 = 0.0015;
%r= 0.00001 + (0.0075-0.00001).*rand; % random generator for the
%Resistance
%r1= r;  r2= r;  r3= r;  r4=r;

%The Mass Values (which ranges from 0.0001 - 0.01)
m1 = 0.001; m2 = 0.001; m3 = 0.001; m4 = 0.001;
%m= 0.0001 + (0.01-0.0001).*rand;  % random generator for the masses
%m1= m;  m2= m;  m3= m;  m4=m;

%The Min Elastance Values (which ranges from 1 - 5)
e1min = 1;    e2min = 1;    e3min = 1;    e4min = 1;
%emin= 1 + (5-1).*rand; % random generator for the
%Min Elastance
%emin1= emin;  emin2= emin;  emin3= emin;  emin4=emin;

%The Max Elastance Values (which ranges from 10 - 50)
e1max = 20; e2max = 40; e3max = 20; e4max = 20;
%emax= 10 + (50-10).*rand; % random generator for the
%Max Elastance
%emax1= emax;  emax2= emax;  emax3= emax;  emax4=emax;

%Heart Perimeters
bpm = 60;   step = 0.001;   cycle = 5;
```

D.2.16. Scripted_Initialization_all_dyss_region2.m

```
% Scripted Initialization Functions
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max e3max e4max bpm
step_size cycle t

%The Resistance Values (which ranges from 0.00001 - 0.0075)
r1 = 0.0015;  r2 = 0.0075;  r3 = 0.0015;  r4 = 0.0015;
%r= 0.00001 + (0.0075-0.00001).*rand; % random generator for the
%Resistance
%r1= r;  r2= r;  r3= r;  r4=r;

%The Mass Values (which ranges from 0.0001 - 0.01)
m1 = 0.001; m2 = 0.01; m3 = 0.001; m4 = 0.001;
%m= 0.0001 + (0.01-0.0001).*rand;  % random generator for the masses
%m1= m;  m2= m;  m3= m;  m4=m;

%The Min Elastance Values (which ranges from 1 - 5)
e1min = 1;    e2min = 4;    e3min = 1;    e4min = 1;
%emin= 1 + (5-1).*rand; % random generator for the
%Min Elastance
%emin1= emin;  emin2= emin;  emin3= emin;  emin4=emin;

%The Max Elastance Values (which ranges from 10 - 50)
e1max = 20; e2max = 40; e3max = 20; e4max = 20;
%emax= 10 + (50-10).*rand; % random generator for the
%Max Elastance
%emax1= emax;  emax2= emax;  emax3= emax;  emax4=emax;

%Heart Perimeters
bpm = 60;   step_size = 0.001;   cycle = 5;
```

D.2.17. Scripted_Initialization_m2_dyss_0_01.m

```
% Scripted Initialization Functions
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max e3max e4max bpm
step_size cycle t

%The Resistance Values (which ranges from 0.00001 - 0.0075)
r1 = 0.0015;  r2 = 0.0075;  r3 = 0.0025;  r4 = 0.0035;
%r= 0.00001 + (0.0075-0.00001).*rand; % random generator for the
%Resistance
%r1= r;  r2= r;  r3= r;  r4=r;

%The Mass Values (which ranges from 0.0001 - 0.01)
m1 = 0.001; m2 = 0.01; m3 = 0.005; m4 = 0.0075;
%m= 0.0001 + (0.01-0.0001).*rand;  % random generator for the masses
%m1= m;  m2= m;  m3= m;  m4=m;

%The Min Elastance Values (which ranges from 1 - 5)
e1min = 1;    e2min = 4;    e3min = 3;    e4min = 5;
%emin= 1 + (5-1).*rand; % random generator for the
%Min Elastance
%emin1= emin;  emin2= emin;  emin3= emin;  emin4=emin;

%The Max Elastance Values (which ranges from 10 - 50)
e1max = 20; e2max = 40; e3max = 30; e4max = 50;
%emax= 10 + (50-10).*rand; % random generator for the
%Max Elastance
%emax1= emax;  emax2= emax;  emax3= emax;  emax4=emax;

%Heart Perimeters
bpm = 60;   step_size = 0.001;   cycle = 5;
```

D.3. Sinusoid Model Code

D.3.1. model_sin.m

```
%Peter Hettwer
%Thesis Work
%3 February 2015

%Simplified Model for Testing Optimization Algorithms

%"Walking" Sinusoids
%phase should be a value between 0 and 2pi

%generates two sine waves based on supplied parameters and noise range

function [sin_x, sin_y] = model_sin(seed, t, params, noise, freq_y)

%sets values of reference sinusoid
phase_x = 2;
amp_x = 1;
freq_x = 1;

%Set default values for experimental parameters if not suplied.
if (nargin < 5)
   freq_y = freq_x;
end
if (nargin < 4)
   noise = 0.1;
end
if (nargin < 3)
   phase_y = -pi + (pi - (-pi))*rand;
end
if (nargin < 2)
   t = linspace(0, 1, 1000);
end
if (nargin < 1)
   seed = 0;
end

amp_y = params(2);
phase_y = params(1);

%create noise profiles for each sinusoid randomly based on supplied noise
%standard deviation
noise_x = 1 + noise*randn(1,length(t));
noise_y = 1 + noise*randn(1,length(t));
```

```
%Create reference waveform.
w_x = 2*pi*freq_x;
sin_x = amp_x * sin(w_x.*t + phase_x) + noise_x;

%Create test waveform (modified by inputs to model by optimization
%algorithm).
w_y = 2*pi*freq_y;
sin_y = amp_y * sin(w_y.*t + phase_y) + noise_y;
end
```

D.3.2. model_sin_plot.m

```
%Peter Hettwer
%Thesis Work
%3 February 2015

%Plot sinusoids based on supplied characteristics


function model_sin_plot(t, phase_y, amp_y, noise_x, noise_y)


%initialize reference sinusiod parameters
phase_x = 2;
amp_x = 1;
freq_x = 1;
freq_y = freq_x;

%find min, max, and mean values for supplied parameter data sets
[min_phase_y, index_min_py] = min(phase_y);
[max_phase_y, index_max_py] = max(phase_y);
mean_phase_y = mean(phase_y);
[min_amp_y, index_min_ay] = min(amp_y);
[max_amp_y, index_max_ay] = max(amp_y);
mean_amp_y = mean(amp_y);

%Create reference waveform with noise.
w_x = 2*pi*freq_x;
sin_x = amp_x * sin(w_x.*t + phase_x) + noise_x;

%Create test waveforms (modified by inputs to model by optimization
%algorithm). Show waveforms based on searches for min/max values for each
%phase and amplitude and a final waveform based on the average values
%computed from each phase and amplitude
w_y = 2*pi*freq_y;

%sinusoid with noise created from data in set with minimum phase
sin_y_p_min = amp_y(index_min_py) * sin(w_y.*t + min_phase_y) + noise_y;
%sinusoid with noise created from data in set with maximum phase
sin_y_p_max = amp_y(index_max_py) * sin(w_y.*t + max_phase_y) + noise_y;
%sinusoid with noise created from data in set with minimum amplidude
sin_y_a_min = min_amp_y * sin(w_y.*t + phase_y(index_min_ay)) + noise_y;
%sinusoid with noise created from data in set with maximum amplitude
sin_y_a_max = max_amp_y * sin(w_y.*t + phase_y(index_max_ay)) + noise_y;
```

```
%sinusoid with noise created from averaged amplitude and phase data
sin_y_avg = mean_amp_y * sin(w_y.*t + mean_phase_y) + noise_y;

%reference sinusoid without noise
sin_x_2 = amp_x * sin(w_x.*t + phase_x);

%plot of noisy sinusoids
figure
plot(t,sin_x,'.b',t,sin_x_2,'g');
legend('sample noisy signal','ideal',1);
xlabel('time');
ylabel('amplitude');
title('Sample of signal corrupted by noise');

sin_x = amp_x * sin(w_x.*t + phase_x);
%sinusoid without noise created from data in set with minimum phase
sin_y_p_min = amp_y(index_min_py) * sin(w_y.*t + min_phase_y);
%sinusoid without noise created from data in set with maximum phase
sin_y_p_max = amp_y(index_max_py) * sin(w_y.*t + max_phase_y);
%sinusoid without noise created from data in set with minimum amplidude
sin_y_a_min = min_amp_y * sin(w_y.*t + phase_y(index_min_ay));
%sinusoid without noise created from data in set with maximum amplitude
sin_y_a_max = max_amp_y * sin(w_y.*t + phase_y(index_max_ay));
%sinusoid without noise created from averaged amplitude and phase data
sin_y_avg = mean_amp_y * sin(w_y.*t + mean_phase_y);

%plot of sinusiods without noise
figure
plot(t,sin_x,'.b',t,sin_y_p_min,'g',t,sin_y_p_max,'r',t,sin_y_a_min,'k',t,sin_y_a_max,'c',t,sin_y_av
g,'.g');
legend('ideal','min phase','max phase','min ampl','max ampl','avg',1);
xlabel('time');
ylabel('amplitude');
title('GA Solution vs. Ideal Solution shown w/o Noise');
end
```

APPENDIX E. EXHAUSTIVE SEARCH CODE

E.1. Exhaustive Search: 2 Parameter

Exhaustive_Search_CD_A.m

```
%Peter Hettwer
%Exhaustive Search Method
%For use with CD Model, 2 parameter (1 reference, 1 search)

%Code to execute exhaustive search operations for the various 2 parameter
%dyssynchrony sets shown in the corrisponding thesis appendix section.

%Prior to execution, user or user generated script file should open a
%parallel MATLAB workstation with the following command:
%matlabpool('open',4); where the number is the number of MATLAB cores
%desired to run.  Maximum Number of cores is dictated by the number of
%physical cores on the computer running MATLAB.  After execution, the user
%or a user generated script file should close the parallel MATLAB
%workstation with the following command: matlabpool('close').

function [] = Exhaustive_Search_CD_A(file, num_points, noise)
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max...
    e3max e4max bpm step_size cycle t

tic; %start MATLAB timer to measure how long search takes

%Scripted_Initialization; %this CD model parameter initialization file is
%needed if this code is not called from a script file previously calling
%the initialization file.

ti = 0:step_size:1; %set period time values based on desired step size;
%used to create acceleration wave plots.

cost = zeros(1,num_points); %create vector of cost values of length
%determined at run-time by user

%Vector of region 1 acceleration wave forms
A1 = zeros(num_points, length(ti));
%Vector of region 2 acceleration wave forms
A2 = zeros(num_points, length(ti));
%Vector of region 3 acceleration wave forms
A3 = zeros(num_points, length(ti));
%Vector of region 4 acceleration wave forms
A4 = zeros(num_points, length(ti));
```

```
p1 = 1; %region 1 timing is assumed as reference,
p2 = linspace(0,2*pi,num_points); %region 2 is being searched
p3 = p1;  %region 3 is tied to region 1
p4 = p1;  %region 4 is tied to region 1


parfor k = 1:num_points; %Parallel for loop to speed up code execution
   %compute acceleration wave forms for regional timing values
   [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1,p2(k),p3,...
      p4,step_size,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,...
      e3min,e3max,e4min,e4max,m1,m2,m3,m4);
   %compute cost value for current set of acceleration wave forms
   cost(k) = SSE(A1(k,:), A2(k,:));
end

%find minimum cost value and index value in region 2
[min_cost index] = min(cost);
p2_best_fit = p2(index) %find the best value for region 2 timing


run_time = toc %stop MATLAB Timer and display to command window

save(file); %save to user defined filename

end
```

E.2. Exhaustive Search: 4 Parameter

Exhaustive_Search_CD_4param_A.m

```
%Peter Hettwer
%Exhaustive Search Method
%For use with CD Model, 2 parameter (1 reference, 3 search)

%Code to execute exhaustive search operations for the various 4 parameter
%dyssynchrony sets shown in the corrisponding thesis appendix section.

%Prior to execution, user or user generated script file should open a
%parallel MATLAB workstation with the following command:
%matlabpool('open',4); where the number is the number of MATLAB cores
%desired to run.  Maximum Number of cores is dictated by the number of
%physical cores on the computer running MATLAB.  After execution, the user
%or a user generated script file should close the parallel MATLAB
%workstation with the following command: matlabpool('close').

function [] = Exhaustive_Search_CD_4param_A(file, num_points,...
    num_runs, noise)
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max...
    e3max e4max bpm step_size cycle t

tic; %start MATLAB timer to measure how long search takes

%Scripted_Initialization; %this CD model parameter initialization file is
%needed if this code is not called from a script file previously calling
%the initialization file.

ti = 0:step_size:1; %set period time values based on desired step size;
%used to create acceleration wave plots.

cost = zeros(1,num_points); %create cost vector for all cost values over
%range for region 2 for a given value for each region 3 and region 4.

range_min = 0; %minimum starting value for search space
range_max = 2*pi; %maximum starting value for search space

i_low = range_min; %minimum starting value for one searched variable
i_high = range_max; %maximum starting value for one searched variable
j_low = range_min; %minimum starting value for second searched variable
j_high = range_max; %maximum starting value for second searched variable
k_low = range_min; %minimum starting value for third searched variable
k_high = range_max; %maximum starting value for third searched variable
```

```
%Vector of region 1 acceleration wave forms
A1 = zeros(num_points, length(ti));
%Vector of region 2 acceleration wave forms
A2 = zeros(num_points, length(ti));
%Vector of region 3 acceleration wave forms
A3 = zeros(num_points, length(ti));
%Vector of region 4 acceleration wave forms
A4 = zeros(num_points, length(ti));

%initial search ranges
p1 = 1; %region 1 timing is assumed as reference,
p2 = linspace(range_min,range_max,num_points) %region 2 is being optimized
p3 = linspace(range_min,range_max,num_points) %region 3 is being optimized
p4 = linspace(range_min,range_max,num_points) %region 4 is being optimized

for r = 1:num_runs; %implements search 'zoom' feature.
   %Zooms for user selected number of times,

   cost_min = NaN; %resets minumum cost at onset of each zoom cycle

   %creates new min and max search space values if not initial run
   if(r ~= 1)

      %create new p2, p3, and p4 test vectors
      if(index_i==1)
         %if lowest best fit == index 1, keep same lower bound as
         %previous.
         i_low = i_low;
      else
         i_low = p4(index_i-1);
      end

      if(index_j==1)
         %if lowest best fit == index 1, keep same lower bound as
         %previous.
         j_low = j_low;
      else
         j_low = p3(index_j-1);
      end

      if(index_k==1)
         %if lowest best fit == index 1, keep same lower bound as
         %previous.
         k_low = k_low;
      else
         k_low = p2(index_k-1);
```

```
    end

    if(index_i==num_points)
        %if highest best fit == max index value, keep same lower bound
        %as previous.
        i_high = i_high;
    else
        i_high = p4(index_i+1);
    end

    if(index_j==num_points)
        %if highest best fit == max index value, keep same lower bound
        %as previous.
        j_high = j_high;
    else
        j_high = p3(index_j+1);
    end

    if(index_k==num_points)
        %if highest best fit == max index value, keep same lower bound
        %as previous.
        k_high = k_high;
    else
        k_high = p2(index_k+1);
    end

    p2 = linspace(k_low,k_high,num_points)  %new region 2 range
    p3 = linspace(j_low,j_high,num_points)  %new region 3 range
    p4 = linspace(i_low,i_high,num_points)  %new region 4 range

end

for i = 1:num_points; %iterates through region 4 values
    for j = 1:num_points; %itterates through region 3 values
        parfor k = 1:num_points; %parallel itteration through
            %region 2 values
            %compute acceleration wave forms for regional timing values
            [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1,...
                p2(k),p3(j),p4(i),step_size,cycle,bpm,r1,r2,r3,r4,...
                e1min,e1max,e2min,e2max,e3min,e3max,e4min,e4max,m1,...
                m2,m3,m4);
            %compute cost value for current set of acceleration wave
            %forms
            cost(k) = SSE4(A1(k,:), A2(k,:), A3(k,:), A4(k,:));
        end
        %check and record if any newly computed cost values are the new
```

```
            %minimum, replace old minimum value and region indices as
            %needed.
            if((min(cost)<cost_min) || isnan(cost_min))
                [cost_min index_k] = min(cost)
                index_i = i
                index_j = j
%               else
%                   i
%                   j
            end
        end
    end
end

[min_cost index] = min(cost); %record minumum cost as standardized
%variable for plotting scripts
p2_best_fit = p2(index_k) %record best region 2 value as standardized
%variable for plotting scripts
p3_best_fit = p3(index_j) %record best region 3 value as standardized
%variable for plotting scripts
p4_best_fit = p4(index_i) %record best region 4 value as standardized
%variable for plotting scripts

run_time = toc %stop MATLAB Timer and display to command window

save(file); %save to user defined filename

end
```

E.3. Exhaustive Search: 2 Parameter, R2 Cost Curve

Same code as Exhaustive_Search_CD_A.m shown previously.

E.4. Exhaustive Search: 2 Parameter, R3 Cost Curve

Exhaustive_Search_CD_A3.m

```
%Peter Hettwer
%Exhaustive Search Method
%For use with CD Model, 2 parameter (1 reference, 1 search)

%Code to Create the Cost Curve for Region 3 of the CD Model.
%Regions 1, 2, and 4 are held constant and equal.

%Prior to execution, user or user generated script file should open a
%parallel MATLAB workstation with the following command:
%matlabpool('open',4); where the number is the number of MATLAB cores
%desired to run.  Maximum Number of cores is dictated by the number of
%physical cores on the computer running MATLAB.  After execution, the user
%or a user generated script file should close the parallel MATLAB
%workstation with the following command: matlabpool('close').

function [] = Exhaustive_Search_CD_A3(file, num_points, noise)
%global variables as required by CD Model Code
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min ...
    e4min e1max e2max e3max e4max bpm step_size cycle t

tic; %start MATLAB timer to measure how long search takes

%Scripted_Initialization; %this CD model parameter initialization file is
%needed if this code is not called from a script file previously calling
%the initialization file.

ti = 0:step_size:1; %set period time values based on desired step size;
%used to create acceleration wave plots.

cost = zeros(1,num_points); %create vector of cost values of length
%determined at run-time by user

%Vector of region 1 acceleration wave forms
A1 = zeros(num_points, length(ti));
%Vector of region 2 acceleration wave forms
A2 = zeros(num_points, length(ti));
%Vector of region 3 acceleration wave forms
A3 = zeros(num_points, length(ti));
%Vector of region 4 acceleration wave forms
A4 = zeros(num_points, length(ti));

p1 = 1; %region 1 timing is assumed as reference,
```

```
p2 = p1; %region 2 is tied to region 1
p3 = linspace(0,2*pi,num_points);  %region 3 is being searched
p4 = p1; %region 4 is tied to region 1


parfor k = 1:num_points; %Parallel for loop to speed up code execution
   %compute acceleration wave forms for regional timing values
   [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1,p2,p3(k),p4,...
      step_size,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,e3min,...
      e3max,e4min,e4max,m1,m2,m3,m4);

   %compute cost value for current set of acceleration wave forms
   cost(k) = SSE4(A1(k,:), A2(k,:), A3(k,:), A4(k,:));
end

%find minimum cost value and index value in region 3
[min_cost index] = min(cost);

p3_best_fit = p3(index) %find the best value for region 3 timing


run_time = toc %stop MATLAB Timer and display to command window

save(file); %save to user defined filename
end
```

E.5. Exhaustive Search: 2 Parameter, R4 Cost Curve

Exhaustive_Search_CD_A4.m

```
%Peter Hettwer
%Exhaustive Search Method
%For use with CD Model, 2 parameter (1 reference, 1 search)

%Code to Create the Cost Curve for Region 4 of the CD Model.
%Regions 1, 2, and 3 are held constant and equal.

%Prior to execution, user or user generated script file should open a
%parallel MATLAB workstation with the following command:
%matlabpool('open',4); where the number is the number of MATLAB cores
%desired to run.  Maximum Number of cores is dictated by the number of
%physical cores on the computer running MATLAB.  After execution, the user
%or a user generated script file should close the parallel MATLAB
%workstation with the following command: matlabpool('close').

function [] = Exhaustive_Search_CD_A4(file, num_points, noise)
%global variables as required by CD Model Code
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min...
    e4min e1max e2max e3max e4max bpm step_size cycle t

tic; %start MATLAB timer to measure how long search takes

%Scripted_Initialization; %this CD model parameter initialization file is
%needed if this code is not called from a script file previously calling
%the initialization file.

ti = 0:step_size:1; %set period time values based on desired step size;
%used to create acceleration wave plots.

cost = zeros(1,num_points); %create vector of cost values of length
%determined at run-time by user

%Vector of region 1 acceleration wave forms
A1 = zeros(num_points, length(ti));
%Vector of region 2 acceleration wave forms
A2 = zeros(num_points, length(ti));
%Vector of region 3 acceleration wave forms
A3 = zeros(num_points, length(ti));
%Vector of region 4 acceleration wave forms
A4 = zeros(num_points, length(ti));

p1 = 1; %region 1 timing is assumed as reference,
```

```
p2 = p1; %region 2 is tied to region 1
p3 = p1; %region 3 is tied to region 1
p4 = linspace(0,2*pi,num_points); %region 4 is being searched

parfor k = 1:num_points; %Parallel for loop to speed up code execution
   %compute acceleration wave forms for regional timing values
   [A1(k,:), A2(k,:), A3(k,:), A4(k,:)] = Scripted_GUI(p1,p2,p3,p4(k),...
      step_size,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,e2max,e3min,...
      e3max,e4min,e4max,m1,m2,m3,m4);
   %compute cost value for current set of acceleration wave forms
   cost(k) = SSE4(A1(k,:), A2(k,:), A3(k,:), A4(k,:));
end

%find minimum cost value and index value in region 4
[min_cost index] = min(cost);

p4_best_fit = p4(index) %find the best value for region 4 timing


run_time = toc %stop MATLAB Timer and display to command window

save(file); %save to user defined filename

end
```

E.6. Exhaustive Search: 4 Parameter, R3 & R4 Cost Surface

Exhaustive_Search_CD_4param_A34.m

```
%Peter Hettwer
%Exhaustive Search Method
%For use with CD Model, 4 parameter (1 reference, 3 search)

%Code to Create the Cost Surface for varying Regions 3 & 4 of the CD Model.
%Regions 1, and 2 are held constant and equal.

%Prior to execution, user or user generated script file should open a
%parallel MATLAB workstation with the following command:
%matlabpool('open',4); where the number is the number of MATLAB cores
%desired to run.  Maximum Number of cores is dictated by the number of
%physical cores on the computer running MATLAB.  After execution, the user
%or a user generated script file should close the parallel MATLAB
%workstation with the following command: matlabpool('close').

function [] = Exhaustive_Search_CD_4param_A34(file, num_points, ...
    num_runs, noise)
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max ...
    e3max e4max bpm step_size cycle t

tic; %start MATLAB timer to measure how long search takes

%Scripted_Initialization; %this CD model parameter initialization file is
%needed if this code is not called from a script file previously calling
%the initialization file.

ti = 0:step_size:1; %set period time values based on desired step size;
%used to create acceleration wave plots.

range_min = 0; %minimum starting value for search space
range_max = 2*pi; %maximum starting value for search space

i_low = range_min; %minimum starting value for one searched variable
i_high = range_max; %maximum starting value for one searched variable
j_low = range_min; %minimum starting value for second searched variable
j_high = range_max; %maximum starting value for second searched variable
k_low = range_min; %minimum starting value for third searched variable,
%not used for this modified code
k_high = range_max; %maximum starting value for third searched variable,
%not used for this modified code

%Vector of region 1 acceleration wave forms
```

```
A1 = zeros(num_points, length(ti));
%Vector of region 2 acceleration wave forms
A2 = zeros(num_points, length(ti));
%Vector of region 3 acceleration wave forms
A3 = zeros(num_points, length(ti));
%Vector of region 4 acceleration wave forms
A4 = zeros(num_points, length(ti));

%initial search ranges
p1 = 1; %region 1 timing is assumed as reference,
p2 = p1; %region 2 is tied to region 1
p3 = linspace(range_min,range_max,num_points) %region 3 is being searched
p4 = linspace(range_min,range_max,num_points) %region 4 is being searched

% cost = zeros(num_points,num_points);
cost = zeros(50,50);

for r = 1:num_runs; %implements search 'zoom' feature.
   %Zooms for user selected number of times,
   %feature not utilized in this search. i.e. num_runs = 1 for creating
   %cost surface

   cost_min = NaN; %resets minumum cost at onset of each zoom cycle

   %creates new min and max search space values if not initial run
   if(r ~= 1)

      %create new p2, p3, and p4 test vectors after initial run
      if(index_i==1)
         %if lowest best fit == index 1, keep same lower bound as
         %previous.
         i_low = i_low;
      else
         i_low = p4(index_i-1);
      end

      if(index_j==1)
         %if lowest best fit == index 1, keep same lower bound as
         %previous.
         j_low = j_low;
      else
         j_low = p3(index_j-1);
      end

      if(index_k==1)
         %if lowest best fit == index 1, keep same lower bound as
```

```
        %previous.
        k_low = k_low;
    else
        k_low = p2(index_k-1);
    end

    if(index_i==num_points)
        %if highest best fit == max index value, keep same lower bound
        %as previous.
        i_high = i_high;
    else
        i_high = p4(index_i+1);
    end

    if(index_j==num_points)
        %if highest best fit == max index value, keep same lower bound
        %as previous.
        j_high = j_high;
    else
        j_high = p3(index_j+1);
    end

    if(index_k==num_points)
        %if highest best fit == max index value, keep same lower bound
        %as previous.
        k_high = k_high;
    else
        k_high = p2(index_k+1);
    end

    p3 = linspace(j_low,j_high,num_points)  %new region 3 range
    p4 = linspace(i_low,i_high,num_points)  %new region 4 range

end

for i = 135:185; %creating surface directly around "best" values,
    %spacing determined using 1001 points between 0 and 2pi
    parfor j = 135:185;
        %compute acceleration wave forms for regional timing values
        [A1(j,:), A2(j,:), A3(j,:), A4(j,:)] = Scripted_GUI(p1,p2,...
            p3(j),p4(i),step_size,cycle,bpm,r1,r2,r3,r4,e1min,e1max,...
            e2min,e2max,e3min,e3max,e4min,e4max,m1,m2,m3,m4);
        %compute cost value for current set of acceleration wave forms
        cost(j,i) = SSE4(A1(j,:), A2(j,:), A3(j,:), A4(j,:));
    end
end
```

```
end

min_cost = min(min(cost)); %determine minimum cost for tested values
[index_j index_i] = find(cost==min_cost); %find indices for tested regions
p2_best_fit = p2 %best value for region 2.  Region 2 was fixed.
p3_best_fit = p3(index_j) %best value for region 3
p4_best_fit = p4(index_i) %best value for region 4

run_time = toc %stop MATLAB Timer and display to command window

save(file); %save to user defined filename

end
```

APPENDIX F. GA SEARCH VARIATIONS FOR WALKING SINUSOID MODEL

F.1. Single Parameter Optimizations

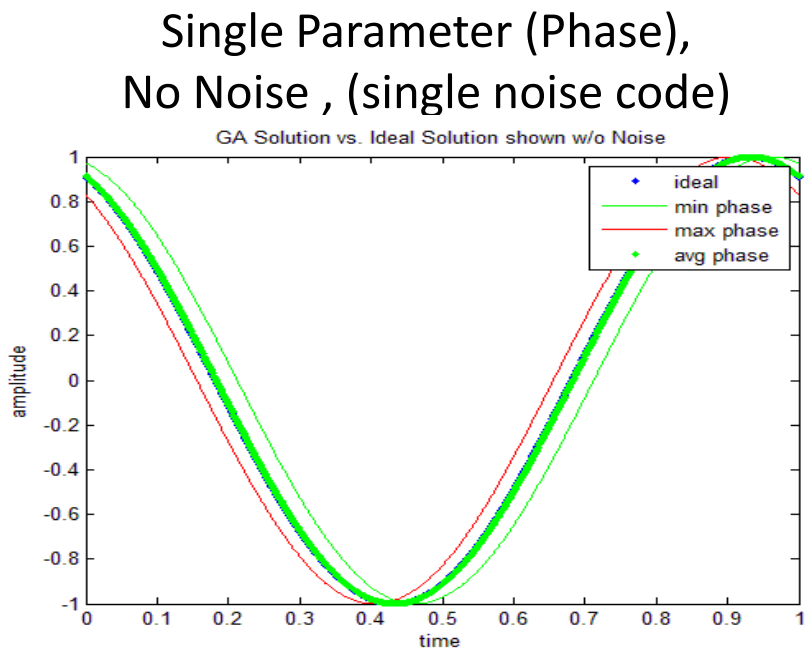

Figure F-1: Single Parameter, No Noise GA Optimization of Walking Sinusoid Model

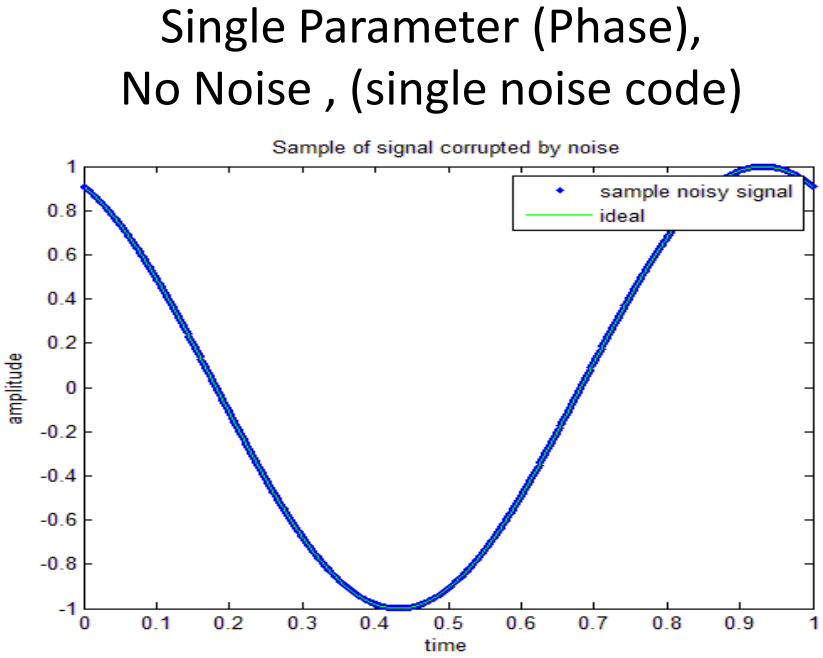

Figure F-2: Noise Level Representation for Single Parameter, No Noise GA Optimization of Walking Sinusoid Model

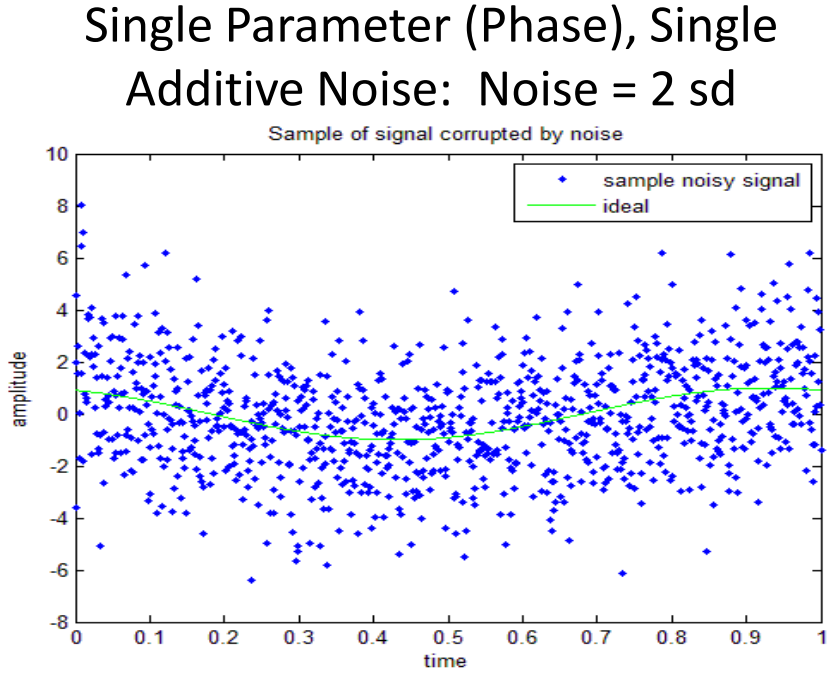

Figure F-3: Noise Level Representation for Single Parameter, 2 Std. Deviation Noise GA Optimization of Walking Sinusoid Model

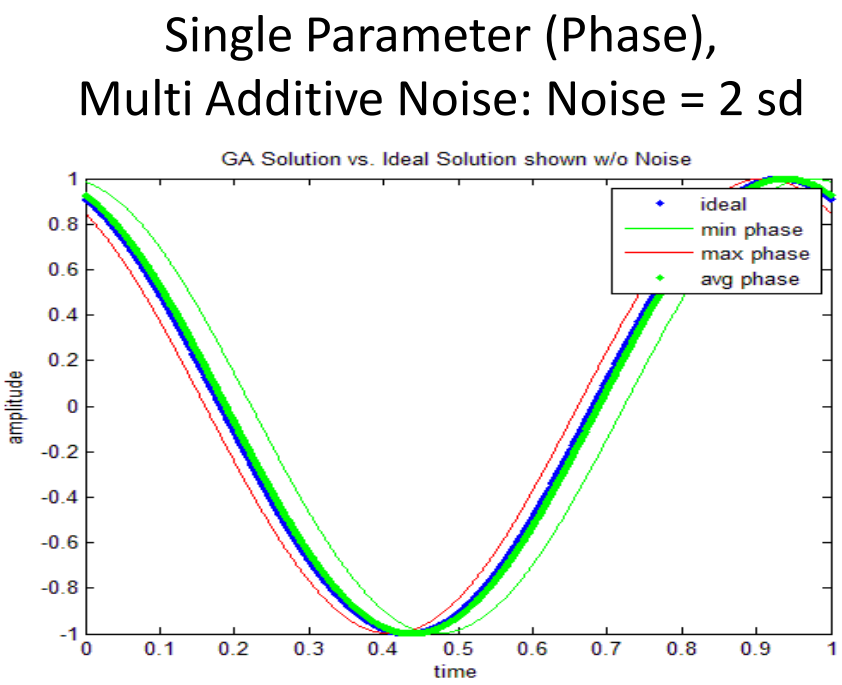

Figure F-4: Single Parameter, 2 Std. Deviation Noise GA Optimization of Walking Sinusoid Model w/ new noise added for each individual comparison to objective function.

F.2. Two Parameter Optimization

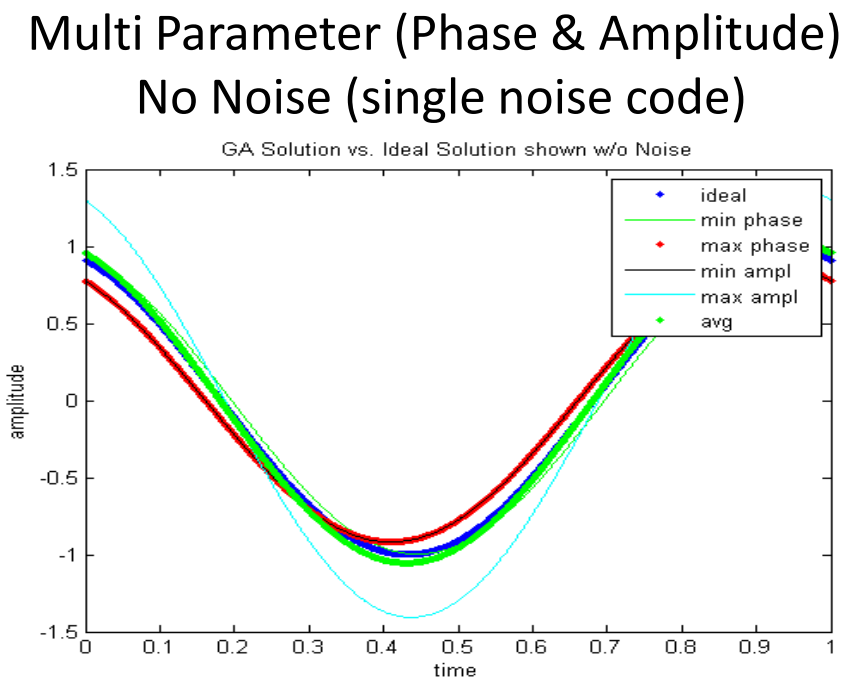

Figure F-5: Two Parameter, No Noise GA Optimization of Walking Sinusoid Model

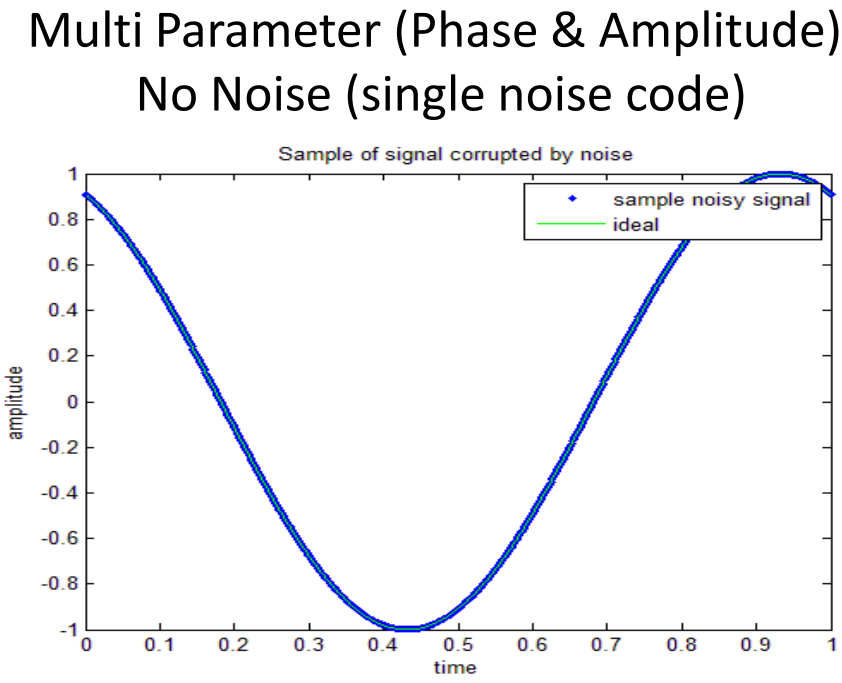

Figure F-6: Noise Level Representation for Two Parameter, No Noise GA Optimization of Walking Sinusoid Model

Multi Parameter (Phase & Amplitude),
Single Additive Noise: Noise = 2 sd

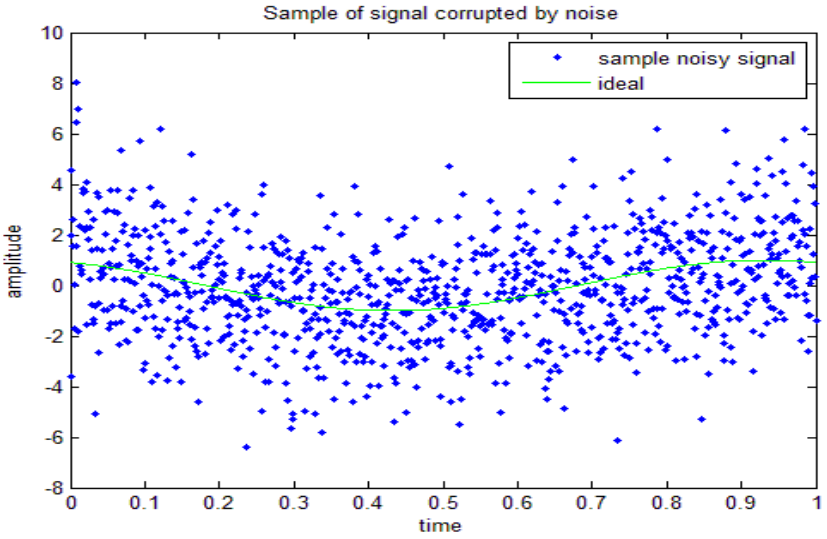

Figure F-7: Noise Level Representation for Two Parameter, 2 Std. Deviation Noise GA Optimization of Walking Sinusoid Model

Multi Parameter (Phase & Amplitude),
Multi Additive Noise: Noise = 2 sd

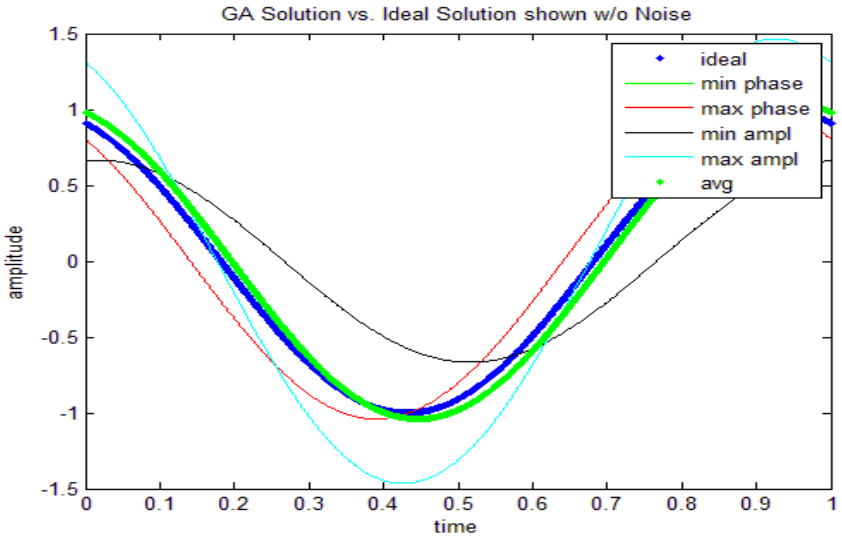

Figure F-8: Two Parameter, 2 Std. Deviation Noise GA Optimization of Walking Sinusoid Model w/ new noise added for each individual comparison to objective function.

APPENDIX G. CARDIAC DYSSYNCHRONY MODEL CODE[1]

The CD model is not strictly clinical in nature; however, that being said, it does provide a stepping stone by which to judge the merits of different optimization algorithms and schemes prior to using more advanced, and computationally intensive models.

[1] The cardiac dyssynchrony model shown in this appendix was originally developed by a team lead by Dr. Dan Ewert, professor of electrical and computer engineering of NDSU specializing in cardiovascular engineering and research, and consisted of Sam Oguyemi, McNair scholar and undergraduate student of mechanical engineering of NDSU, and a group of senior design students at Iron Range Engineering for the purpose of having a graphical user representation of CD for use as a visual aid in classroom instruction on CD and creating an objective set of metrics to measure CD. Assisted by Sam Oguyemi, Peter Hettwer created the shell function Scripted_GUI.m and the parameter initialization file Scripted_Initialization.m to allow model operation without a user interactive graphical user interface (GUI) and accept four relative firing phases from an optimization algorithm after which, it calculates and returns the accelerations of the four left ventricular regions for use in the optimization algorithm creating a closed loop test platform. Peter Hettwer then used the CD model with the optimization algorithms he designed to obtain the data utilized in this thesis.

G.1. Scripted_GUI.m

This code is the base function called within an optimization algorithm to generate regional acceleration waveforms from generated regional phase timings.

```
% Script GUI
function [A1,A2,A3,A4] = Scripted_GUI(P1,P2,P3,P4,step_size,cycle,bpm,...
   r1,r2,r3,r4,e1min,e1max,e2min,e2max,e3min,e3max,e4min,e4max,m1,m2,...
   m3,m4)
%global t1 t2 t3 t4

%Calling the Initialization Script

%step = 1/1000;
%Phase Transformation
t1 = ((1/(bpm/60))/(2*pi))*P1;
t2 = ((1/(bpm/60))/(2*pi))*P2;
t3 = ((1/(bpm/60))/(2*pi))*P3;
t4 = ((1/(bpm/60))/(2*pi))*P4;

% Caling the Dyss Function
[I1,I2,I3,I4] = dyss1(step_size,cycle,bpm,r1,r2,r3,r4,e1min,e1max,e2min,...
   e2max,e3min,e3max,e4min,e4max,m1,m2,m3,m4,t1,t2,t3,t4);

%The Acceleration Values
Aa1 = nineptder1(I1,step_size);
Aa2 = nineptder1(I2,step_size);
Aa3 = nineptder1(I3,step_size);
Aa4 = nineptder1(I4,step_size);
%Method of extrating the last elements at equilibrium
period_length = 1/step_size;
A1 = Aa1(length(Aa1)-period_length:length(Aa1));
A2 = Aa2(length(Aa2)-period_length:length(Aa2));
A3 = Aa3(length(Aa3)-period_length:length(Aa3));
A4 = Aa4(length(Aa4)-period_length:length(Aa4));
```

G.2. Scripted_Initialization.m

This function declares and initializes the values for the mechanical model parameters in each of the four model regions.

```
% Scripted Initialization Functions
global r1 r2 r3 r4 m1 m2 m3 m4 e1min e2min e3min e4min e1max e2max ...
   e3max e4max bpm step_size cycle t

%The Resistance Values (which ranges from 0.00001 - 0.015)
r1 = 0.0015;  r2 = 0.0015;  r3 = 0.0015;  r4 = 0.0015;

%The Mass Values (which ranges from 0.0001 - 0.01)
m1 = 0.001; m2 = 0.001; m3 = 0.001; m4 = 0.001;

%The Min Elastance Values (which ranges from 1 - 5)
e1min = 1;    e2min = 1;    e3min = 1;    e4min = 1;

%The Max Elastance Values (which ranges from 10 - 50)
e1max = 20; e2max = 20; e3max = 20; e4max = 20;

%Heart Perimeters
bpm = 60;   step_size = 0.001;   cycle = 5;
```

G.3. dyss1.m

This file handles model interaction with a MATLAB ordinary differential equation (ode) solver.

```
%Awesome script to run dyssynchrony
function [I1, I2, I3, I4] = dyss1(step,cycle,bpm,r1,r2,r3,r4,e1min,...
   e1max,e2min,e2max,e3min,e3max,e4min,e4max,m1,m2,m3,m4,t1,t2,t3,t4)
%global r1 r2 r3 r4 e1min e1max e2min e2max e3min e3max e4min e4max m1 ...
%m2 m3 m4 t1 t2 t3 t4 step cycle bpm t

t = 0:step:cycle;
min_e = .5;
volume = 10;
pressure = volume*min_e;
OPTIONS=odeset('MaxStep',1e-4);

[a2 b2]=ode23s(@ejection,t,[pressure 0 pressure 0 0 0 pressure  volume ...
   volume 0 0 pressure pressure 0 0 6 100 volume volume min_e r1 r2 r3 ...
   r4 m1 m2 m3 m4],OPTIONS,[r1 r2 r3 r4 e1min e1max e2min e2max e3min ...
   e3max e4min e4max m1 m2 m3 m4 t1 t2 t3 t4  bpm]);


pe1= b2(:,1);
I1 = b2(:,2);
pe2= b2(:,3);
I2 = b2(:,4);
Ii = b2(:,5);
Io = b2(:,6);
LVP= b2(:,7);
V1 = b2(:,8);
V2 = b2(:,9);
Vi = b2(:,10);
Vo = b2(:,11);
pe3= b2(:,12);
pe4= b2(:,13);
I3 = b2(:,14);
I4 = b2(:,15);
LAP= b2(:,16);
AoP= b2(:,17);
V3 = b2(:,18);
V4 = b2(:,19);
W  = b2(:,20);
R1 = b2(:,21);
R2 = b2(:,22);
R3 = b2(:,23);
R4 = b2(:,24);
```

```
M1 = b2(:,25);
M2 = b2(:,26);
M3 = b2(:,27);
M4 = b2(:,28);

assignin('base', 's', step);
t = t;
```

G.4. ejection.m

This function defines the differential equations used within the MATLAB ode solver.

```
%This is ejecting stacked model

function [dy] = ejection(t,y,z)
global dI1
%defining variables
pe1 = y(1);
I1 = y(2);
pe2 = y(3);
I2 = y(4);
Ii = y(5);
Io = y(6);
LVP = y(7);
V1 = y(8);
V2 = y(9);
Vi = y(10);
Vo = y(11);
pe3 = y(12);
pe4 = y(13);
I3 = y(14);
I4 = y(15);
LAP = y(16);
AoP = y(17);
V3 = y(18);
V4 = y(19);
W = y(20);
R1= y(21);
R2= y(22);
R3= y(23);
R4= y(24);
M1 = y(25);
M2 = y(26);
M3 = y(27);
M4 = y(28);
%E1 = y(29);
%E2 = y(30);
%E3 = y(31);
%E4 = y(32);

%resistance dyssynchrony for each section of the heart
r1 = z(1);
r2 = z(2);
r3 = z(3);
```

```
r4 = z(4);

%elastance dyssynchrony for each section of the heart
e1min=z(5);
e1max=z(6);
e2min=z(7);
e2max=z(8);
e3min=z(9);
e3max=z(10);
e4min=z(11);
e4max=z(12);

% e1= e1max-e1min;
% e2= e2max-e2min;
% e3= e3max-e3min;
% e4= e4max-e4min;
%mass dyssynchrony for each section of the heart
m1 = z(13);
m2 = z(14);
m3 = z(15);
m4 = z(16);

%timing dyssynchrony for each section of the heart
t1 = z(17);
t2 = z(18);
t3 = z(19);
t4 = z(20);

%heart beats per minute
bpm = z(21);

% Defining constants
k1 = r1;        %This is really resistance 1
k2 = r2;        %This is really resistance 2
k3 = r3;        %This is really resistance 3
k4 = r4;        %This is really resistance 4
Ri = .01;
Ro = .01;
mi = .0002;
mo = .0002;
%LAP = 10;
%AoP = 100;
Clvp = .0001;
m1 = m1;
m2 = m2;
m3 = m3;
```

```
m4 = m4;
Rs = 2.5;
Cv = 100;
Ca = 20;

% Attain time varying parameters
[e1 de1] = getk(t+t1,e1min,e1max,bpm);
[e2 de2] = getk(t+t2,e2min,e2max,bpm);% for dyssynchrony create second offset elastance
[e3 de3] = getk(t+t3,e3min,e3max,bpm);
[e4 de4] = getk(t+t4,e4min,e4max,bpm);

% Writing differential equations
dpe1 = e1*(I1+pe1*(1/(e1)^2*de1)); % next four for elastance
%dpe2 = e1*(I2+pe2*(1/e1)^2*de1);
dpe2 = e2*(I2+pe2*(1/e2)^2*de2); % switch percent to the one above for dyssnchrony
dpe3 = e3*(I3+pe3*(1/e3)^2*de3);
dpe4 = e4*(I4+pe4*(1/e4)^2*de4);

dI1 = (1/m1)*(LVP-pe1-(k1*LVP)*I1); % flow for next four for mass
dI2 = (1/m2)*(LVP-pe2-(k2*LVP)*I2);
dI3 = (1/m3)*(LVP-pe3-(k3*LVP)*I3);
dI4 = (1/m4)*(LVP-pe4-(k4*LVP)*I4);

%  if (LVP>AoP)
%     Ro=.1;
%   else
%      Ro=1e3;
%   end
Do=20*(-(.15/(.15+exp(-6*Io)))+1);

dIo = (1/mo)*(LVP-AoP-(Ro+Do)*Io); % next two for flow in/flow out

%   if (LAP>LVP)
%      Ri=.1;
%   else
%      Ri=1e3;
%   end

Di=20*(-(.15/(.15+exp(-6*Ii)))+1);

dIi = (1/mi)*(LAP-LVP-(Ri+Di)*Ii);

Ilvp = Ii-Io-I1-I2-I3-I4; % flow balance
dLVP = (1/Clvp)*Ilvp; % for the capacitor

dLAP=(Ii-((AoP-LAP)/Rs))/-Cv;
```

```
dAoP=(Io-((AoP-LAP)/Rs))/Ca;

%w = LVP*(V1+V2+V3+V4)     %work
dW = LVP*(I1+I2+I3+I4);

%Elastic power for each heart segment (ideal)
% dwe1=-e1*V1*I1*.00013;
% dwe2=-e2*V2*I2*.00013;
% dwe3=-e3*V3*I3*.00013;
% dwe4=-e4*V4*I4*.00013;

%Power disipated by myocardial resistance (dash pot)
dwr1=k1*LVP*I1^2*.00013;
dwr2=k2*LVP*I2^2*.00013;
dwr3=k3*LVP*I3^2*.00013;
dwr4=k4*LVP*I4^2*.00013;

%Derivative of the Power disipated by myocardial resistance
%ewr1=[0 diff(dwr1)];
%ewr2=[0 diff(dwr2)];
%ewr3=[0 diff(dwr3)];
%ewr4=[0 diff(dwr4)];

%Kinetic power of the mass of the heart
dwk1=m1*I1*9.81*10^-5;
dwk2=m2*I2*9.81*10^-5;
dwk3=m3*I3*9.81*10^-5;
dwk4=m4*I4*9.81*10^-5;

%w1 = (Vi*Ii^2/2 + Vi*(LAP-Ri*Ii) - Vo*Io^2/2 - Vo*(AoP-Ro*Io))/(V1+V2)

dy =
[dpe1;dI1;dpe2;dI2;dIi;dIo;dLVP;I1;I2;Ii;Io;dpe3;dpe4;dI3;dI4;dLAP;dAoP;I3;I4;de1;dwr1;dwr
2;dwr3;dwr4;dwk1;dwk2;dwk3;dwk4]; %ewr1;ewr2;ewr3;ewr4];
%The dy Solves for the integral of the inputted variables

%Q= diff (R1);
end

% function setGlobalAa1(dI1)
% global Aa1
% Aa1=dI1
```

G.5. getk.m

This function generates elastance waveforms from supplied min and max elastance values.

```
function [k,dk] = getk(t,Emin,Emax,bpm)

% global beat
% global t_old
t = mod(t,60/bpm);

a=1; %scales normal distribution to 1
b=.5*60/bpm; % centers the mean at 1/2 of the cycle
c=.23*b; %makes the spread of curve to 50% duty cycle
k=(Emax-Emin)*a*exp(-(t-b).^2 /(2*c.^2))+Emin;
dk=(Emax-Emin)*a*exp(-(t-b).^2 /(2*c.^2)).*(-2*(t-b)/(2*c.^2));

% % create elastance waveform
% t = mod(t,60/bpm);
% q = (Emax-Emin)*sin(2*pi*t*(60/bpm));
% w = .5*square(2*pi*t*(60/bpm)) + .5;
% k = q.*w + Emin;
% % t
%
% % create derivative of elastance waveform
%
%
% % this creates 5 points centered on k, and then offset by k-2, k-1, k+1,
% % and k+2...   Add these together with weighting function to get 5 pt der.
%
% dq = 2*pi*(60/bpm)*(Emax-Emin)*cos(2*pi*t*(60/bpm));
% dw = .5*square(2*pi*t*(60/bpm)) + .5;
% dk = dq.*dw;

end
```

G.6. nineptder1.m

This function computes specific derivatives for various data vectors within the model.

```
function [dxdt]=nineptder1(x,dt)

% Computes the derivative using algoritm from Numerical Analysis, 2ed.,
% Burden, Faires, Reynolds; pg.130
% fiveptder(x,fs)
% x is the input signal for derivative calculation
% fs is the sampling frequency
% computes numerical derivative dx/dt

%dt = 1/dt;
%We need the time derivative of AoF so what follows is a numerical
%5-point derivative
dxdt=ones(length(x),1);
a=length(x);


dxdt(1)=(x(2)-x(1))/dt;
dxdt(2)=(x(3)-x(2))/dt;
dxdt(a-1)=(x(a-1)-x(a-2))/dt;
dxdt(a)=(x(a)-x(a-1))/dt;

% 5 point Derivative for the third and fourth points
for i = 3:4
dxdt(i)=[1/(12*dt)] * [x(i-2) - 8*x(i-1) + 8*x(i+1) - x(i+2)];
end

% 5 point Derivative for the third and fourth points from the end
for i = a-3:a-2
dxdt(i)=[1/(12*dt)] * [x(i-2) - 8*x(i-1) + 8*x(i+1) - x(i+2)];
end


% 9 point Derivative
for i = 5:length(x)-4
dxdt(i)=[1/(840*dt)] * [3*x(i-4) - 32*x(i-3) + 168*x(i-2) - 672*x(i-1)...
   + 672*x(i+1) - 168*x(i+2) + 32*x(i+3) - 3*x(i+4) ];
% previous algoritm is from Numerical Analysis, 2ed., Burden, Faires,
% Reynolds; pg.130
end
```