Cleveland State University
EngagedScholarship@CSU

MSL
ACADEMIC ENDEAVORS

ETD Archive

2014

Biogeography-Based Optimization of a Variable Camshaft Timing System

George L. Thomas
Cleveland State University

BIOGEOGRAPHY-BASED OPTIMIZATION OF A VARIABLE CAMSHAFT TIMING SYSTEM

GEORGE THOMAS

Bachelor of Electrical Engineering
Cleveland State University
December, 2012

submitted in partial fulfillment of requirements for the degree
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
at the
CLEVELAND STATE UNIVERSITY
December, 2014

We hereby approve the thesis of

George Thomas

Candidate for the Master of Science in Electrical Engineering degree.

This thesis has been approved for the department of

ELECTRICAL AND COMPUTER ENGINEERING

and

CLEVELAND STATE UNIVERSITY

College of Graduate Studies by

____

Thesis Committee Chairperson, Dr. Dan Simon

____

Department of Electrical and Computer Engineering, 11/24/2014

____

Dr. Zhiqiang Gao

____

Department of Electrical and Computer Engineering, 11/24/2014

____

Dr. Mehdi Jalalpour

____

Department of Civil and Envronmental Engineering, 11/24/2014

Student's Date of Defense: 11/24/2014

ACKNOWLEDGMENTS

I would like to thank the Ford Motor Company for financially supporting this research. I also thank my colleagues from Ford, John Michelini and Dimitar Filev, who have helped shape this thesis. I thank Cleveland State University (CSU) for its financial support throughout this thesis and the rest of my education and Dr. Dan Simon for his help and guidance over the six years I’ve spent there. I thank Dr. Zhiqiang Gao for inspiring me to foster a mindful and inquisitive approach to engineering, and I thank Dr. Mehdi Jalalpour his help and willingness to serve on this committee without meeting me first. I also thank Rick Rarick, Ron Davis, and the many others who I’ve worked with as a research assistant at CSU. I am happy to call them friends as well as coworkers. Finally, I thank my fiancée, family, and friends who have supported me with sorely needed help and love and encouragement throughout my entire education. I cannot thank them enough.

BIOGEOGRAPHY-BASED OPTIMIZATION OF A VARIABLE CAMSHAFT TIMING SYSTEM

GEORGE THOMAS

ABSTRACT

Automotive system optimization problems are difficult to solve with traditional optimization techniques because the optimization problems are complex, and the simulations are computationally expensive. These two characteristics motivate the use of evolutionary algorithms and meta-modeling techniques respectively. In this work, we apply biogeography-based optimization (BBO) to radial basis function (RBF)-based lookup table controls of a variable camshaft timing system for fuel economy optimization. Also, we reduce computational search effort by finding an effective parameterization of the problem, optimizing the parameters of the BBO algorithm for the problem, and estimating the cost of a portion of the candidate solutions in BBO with design and analysis of computer experiments (DACE). We find that we can improve fuel economy by 1.7% compared to the original control parameters, and we find effective, problem-specific values for BBO population size and mutation rate. Finally, we find that we can use a small number of samples to construct DACE models, and we can use these models to estimate a significant portion of the BBO candidate solutions each generation to reduce computation effort and still obtain good BBO solutions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

BBO	Biogeography-Based Optimization
DACE	Design and Analysis of Computer Experiments
DOE	Design of Experiments
EA	Evolutionary Algorithm
ICE	Internal Combustion Engine
RBF	Radial Basis Function
VVA	Variable Valve Actuation
VCT	Variable Camshaft Timing

CHAPTER I

INTRODUCTION

1.1 Overview

Internal combustion engine (ICE) variables have been optimized for over half a century, although the optimization approaches and objectives have changed over the years in response to the changing environment in which these engines are used. Early work in ICE optimization includes an application of control systems theory in which the spark timing and air intake throttle are manipulated to optimize the output brake mean effective pressure (BMEP) [1]. The oil embargo to the USA in 1973 [2] and the subsequent corporate average fuel consumption (CAFE) standards [3] set by the US government shifted the auto industry's focus toward optimizing ICEs for fuel economy, instead of engine power output as in [1]. The reduction of exhaust emissions became an objective at about the same time, starting with 1966 regulations in California [4]. Further, the introduction of the microprocessor for engine control in the middle 1970s greatly simplified tuning of engine performance over previous analog, mechanical engine control, and thus facilitated further ICE optimization [4].

Since the 1970s, there has been an explosion in the application of new engineering techniques such as standard and multi-objective evolutionary algorithms (EAs) [5], [6] and artificial neural network-based surrogate models [7] to the optimization of different ICE actuator variables such as variable valve actuation (VVA), spark angle, and air-fuel (A/F) ratio [5], [8] for fuel economy and emissions.

EAs are robust global optimizers and require only cost function evaluations of candidate solutions, instead of requiring differentiable or analytical cost functions. These properties make EAs attractive options for complex, nonlinear, multimodal, or black-box optimization problems like ICE optimization, and thus we have chosen an EA called biogeography-based optimization (BBO) for this research.

1.2 Evolutionary Algorithms

EAs generally work by evolving several different candidate solutions to a given problem in parallel. EAs are most often used for parameter optimization problems, and so candidate solutions are usually sets of parameters that solve these optimization problems. Each step of the evolution process in an EA is called a generation, and the group of candidate solutions is often called a population. During each generation, the EA applies one or more evolutionary operators to the candidate solutions. These evolutionary operators, which are heuristics modeled on different evolutionary processes in nature (e.g., natural selection, mutation, and swarm behavior [9]) are used to produce new candidate solutions using information from the old candidate solutions. New information may also be brought into the population, and the mutation operator found in many EAs is an example of a heuristic used to accomplish this.

These operators often use the cost or fitness of these solutions as inputs to determine how the features of solutions are used (a solution feature may be a value for one of a problems' parameters). Cost is a way of describing the performance of solutions in the context of engineering problem optimization. Good solutions to optimization problems have low costs, and so the objective of these problems is to minimize cost. Fitness is often used to describe solution performance in the context of the theory of an EA, and the optimization objective is to maximize fitness, because good solutions are ones with high fitness. Cost and fitness can both be used to quantify the performance of solutions, but because we are primarily concerned with our optimization problem rather than the theory of a specific EA, we use the term cost to avoid abstracting the discussion too far away from our automotive problem. Also, we often refer to candidate solutions as solutions throughout this work; distinctions between a candidate solution of an EA and a solution in another sense should be clear within context.

One drawback of EAs is their computational complexity. Automotive simulations are often computationally expensive, and if they are to be used as part of a cost function for an EA, they must be run many times. This fact poses a problem of particularly high computational effort. There are several approaches to reducing computational effort for problems that involve computationally expensive cost functions, including keeping track of evaluated candidate solutions to avoid recomputing their cost; truncating cost function evaluations during a run when it can be determined that the solution being evaluated will have a very large or very small cost value; parallelizing cost function evaluations; speed optimization of the cost function computer code; viewing a cost function as a combination of sub-functions and approximating it with sub-functions; and the use of surrogate models of the cost function [10]. We discuss these methods in more detail in the following subsection.

The choices of EA parameters, such as the parameters of evolutionary operators and population sizes, have an effect on the performance of the EA. For instance, a sensitivity analysis of the parameters of a particular GA on two different scheduling problems shows that some of the GA’s parameters are more important for finding better solutions than others; specifically, crossover rate is the most important for one problem, and population size is the most important for the other [11]. Also, different EA parameters emphasize different aspects of the search procedure—in the case of an evolution strategies algorithm, the parent population size reflects the amount of emphasis given to global search or parallel exploration, whereas the offspring population size reflects exploitation of the information contained in the parent solutions [12].

Given the characteristics of a search space (i.e., the specific problem) and a given EA population, the optimal tradeoffs between exploration and exploitation will be different, and thus the optimal EA parameters will be different. This motivates us to optimize the EA parameters for our particular EA and problem, because doing so allows us to find better solutions with less search effort. It should also be noted that adaptive algorithms have been applied to EA parameters [12]. This can be useful because the optimal values of EA parameters can change over the course of an EA run based on characteristics of the population, such as diversity. We choose to use a manual, suboptimal approach for finding effective EA parameters instead of developing adaptive EA parameter optimization, as the latter approach is primarily effective for problems with dynamic or time-varying cost functions [12], while our cost function is a deterministic simulation.

1.3 Reducing Computational Complexity of EAs

The first method of reducing EA computational effort that we have mentioned, to record the cost of all evaluated solutions, is useful because we can avoid computing the costs of previously evaluated EA solutions by looking up their costs in the record. This is mostly helpful for situations for which we expect to have the same solution in a population over multiple generations, such as when elitism is used. Having a record of all previously evaluated solutions is also useful for generating surrogate models, as it may provide a large pool of solutions from which to draw samples for the surrogate modeling process. We have implemented this feature in our BBO software, and we discuss it in Section 4.4.

We can also minimize the CPU time required to compute the cost function. This may include using more computationally efficient algorithms or reducing the computational overhead of the cost function (e.g., removing debugging code), or using compiler optimizations. Because our automotive simulation is written in Simulink, we removed all of the “display,” “output,” and “to workspace” blocks used for debugging. Our simulation was also improved during development to reduce the aggressiveness of a control system used to simulate the driver’s control of the accelerator and brake pedals. We found that this reduced aggressiveness also reduced the simulation time, and we suspect this is because there is less high-frequency chattering of the simulated pedals, and thus the variable-step ordinary differential equation solver in MATLAB can relax the simulation step size and evaluate fewer steps throughout the course of the simulation. After making both of these changes, the CPU time required to run over the full simulation drive trace (the details of which are described in Sections 2.1 and 2.5) was reduced by more than 50%.

We have also parallelized the cost function evaluations performed each generation. We have evaluated a maximum of 12 candidate solution costs in parallel in this research, and in the case of using 12 parallel cost evaluations, we observe a BBO simulation time of about 10% of the nominal simulation time. This is expected, because there is computational overhead associated with parallelization, and we are only parallelizing the cost function evaluations, not the entire BBO algorithm. Our implementation of cost function parallelization is shown in Section 4.3.

We also approximate a full cost function evaluation by the evaluation of a subset of the cost function. We do this by running the vehicle simulation over only a portion of a given drive trace (i.e., vehicle velocity profile.) We describe this approach in Section 4.2.

Surrogate models or response surfaces are of particular interest, because they are often computationally simpler than the expensive functions that they estimate, and there are a variety of different modeling methods, including those based on neural networks [7], polynomials[10], locally weighted regression [13] and stochastic (e.g., Gaussian) processes [14], [15]. Also, a multi-objective genetic algorithm (GA) has been used with kriging-, RBF-, and neural network-based meta models for parameter design optimization—in this work, meta model-based objective functions were optimized, and robust solutions were found by simultaneously minimizing the variance of the objective function values with respect to changing model parameters [16]. Kriging is a Gaussian process-based metamodeling technique from the geostatistics literature and was originally developed for predicting locations of ore deposits [17]. Design and Analysis of Computer Experiments (DACE) is a kriging-based surrogate modeling method, and is described in Chapter III. We apply DACE to BBO to improve the computational efficiency of search with BBO, and we describe this application in Section 4.5.

1.4 Biogeography-Based Optimization

BBO is an EA inspired by biogeography, which is the study of the migration of species between habitats [18]. The novel evolutionary operator in BBO is migration. With migration, each candidate solution in the population is assigned an immigration rate, λ, and an emigration rate, μ, based on its cost; these migration rates determine the likelihood that a solution will give or receive features from other solutions in the population. In standard BBO, these rates are given by

$$\lambda_i = \frac{f_i}{n} \qquad (1.1)$$

$$\mu_i = 1\text{-}\lambda_i \qquad (1.2)$$

where n is the size of the BBO population, $i \in [1 \ldots \mathrm{n}]$ is a solution index, and f_i is the fitness of the ith solution.

Standard features of BBO also include mutation and elitism. With mutation, new information is incorporated into the population. As we have implemented it, the mutation operator in BBO replaces a solution variable in an individual with a given mutation probability, with a value drawn from a uniform distribution, limited to be within the search range for that variable. The mutation probability is a parameter of BBO that is set before starting a run.

Elitism ensures that the best solutions continue to contribute to the evolutionary process. At the end of each generation, we replace a certain number of the poorest solutions with the best solutions from the previous generation. Also, to keep the population from filling up with duplicates of the elite solutions, we replace any duplicated solutions with solutions whose variables are randomly selected from a uniform distribution within the search range.

We have also implemented worsening probability in BBO. This parameter determines the probability that the changes made to a solution by the application of migration and mutation are kept, when these changes make the solution worse.

The BBO algorithm is outlined in pseudocode in Algorithm 1.

```
for each solution $S_i$
  $S_{old} \leftarrow S_i$
  for each solution feature $x$
    select solution $S_i$ for immigration with probability proportional to $\lambda_i$
    if solution $S_i$ is selected then
      select $S_j$ for emigration with probability proportional to $\mu_j$
      if $S_j$ is selected then
        $S_i(x) \leftarrow S_j(x)$
      end
    end
  next solution feature
  mutate $S_i$ with probability $p_m$
  if cost($S_i$) > cost($S_{old}$)
    replace $S_i$ with $S_{old}$ given probability $p_w$
  end
next solution
```

Algorithm 1: Outline of the BBO algorithm given in a pseudocode form, showing how migration and mutation operations and the worsening rate are applied to a given solution, where p_m is the mutation probability, and p_w is the [19]worsening probability.

We have chosen to use BBO as a typical EA. We justify our use of BBO by noting that it outperforms many EAs on a variety of benchmarks [18], and some promising theoretical research has been done to support it, including an analysis of BBO's migration models [20], a theoretical comparison between BBO and GAs [21], and a probabilistic study of BBO [22]. BBO has also been successfully applied to a variety of nonlinear control problems such as fuzzy logic-based robot path tracking control optimization, open-loop knee prosthesis control, and load dispatching in power systems [23], [24], [25]. Because of this, it is reasonable to expect it to perform as well as any other EA for ICE control optimization.

1.5 Contribution

Our main contribution to the topic of engine control tuning is the development of cam timing control tables that result in a 1.7% improvement in fuel economy over the tables that we started with. This fuel economy improvement has important implications that are explained in Section 7.1.

We consider our research to be novel because we are not aware of any applications of DACE to reduce the computational effort associated with using an EA for optimization in an automotive system; however we note that RBF and kriging-based metamodels have been used for modeling emissions response surfaces before, as indicated in [26].

Also, EAs have previously been used to tune VVA systems. In [27], a multiobjective GA was used to simultaneously optimize diesel engine fuel economy and engine torque by manipulating intake valve timing. This means that the application of BBO to the optimization of a variable camshaft timing (VCT) system is not very novel.

Our approach to use sub-functions of a vehicle drive trace may be new, because we have not seen any examples of decomposition of existing drive traces into sub-functions, or examination of the effect of drive trace quantities such as RMS velocity or RMS acceleration on EA optimization performance. We do note that combinations of random drive traces were used in work where stochastic dynamic programming was applied to the optimization of hybrid electric vehicle control [28].

Other work that has already been done includes derivative-based optimization of dual independent cam phasing using high-fidelity engine simulation models [29]. In another work, high fidelity engine simulations were used to train neural network-based surrogate models, which were then optimized to find optimal set points for a variable valve timing system [7]. Also, in [26], turbo-diesel engine control parameters were tuned to simultaneously minimize particulate, NOx, and CO emissions, and minimize engine noise, with a constraint on fuel economy. The algorithm used in this case is a multiobjective evolutionary strategies algorithm. These works exemplify the trend in automotive engineering of the use of simulation model-based engine control optimization, where an increasing variety of engine subsystems are taken into account.

1.6 Organization

The remainder of this paper is organized as follows. In Chapter II, we provide a description of the physical system that we are simulating, and we explore our formulation of the problem and how we choose to construct a parameterization for use with BBO. In Chapter III, we introduce the DACE algorithm, discuss how it works, and describe some potential numerical problems that one may encounter when using it. In Chapter IV, we discuss our implementation of the BBO algorithm, including how we use cost function parallelization, recordkeeping of the cost values of solutions to avoid re-computing them, selection of simulation drive traces, and our application of DACE to reduce computational effort. In Chapter V, we discuss our tuning of BBO and DACE parameters for the cam timing optimization problem; these variables include population size, mutation rate, worsening rate, number of DACE samples, and percentage of individuals per generation to estimate using DACE. In Chapter VI, we examine BBO solutions to the VCT problem in the problem domain, and attempt to explain the qualities that make our solutions perform well. Finally, we draw conclusions and suggest future work in Chapter VII.

CHAPTER II

PROBLEM FORMULATION

2.1 Overview of Vehicle Simulation

The Simulink® vehicle simulation that we use throughout this work is of a passenger vehicle equipped with a turbocharged gasoline powered direct injection engine, and includes variable timing of both intake and exhaust camshafts. Although there are a variety of control systems that we can optimize in this vehicle, we restrict our attention to the VCT system.

VCT is a particular type of VVA technology in which the timing of intake valves, exhaust valves, or both are manipulated by changing the angular position of one or more camshafts with respect to the crankshaft. We have chosen to optimize the only VCT system because we have identified this system as one where large fuel economy improvements can be made with minimal engineering effort. We could optimize the vehicle hardware and find large improvements, but this would be costly. Also, the other controllers of the vehicle have been studied extensively and are better understood, and likely have less room to improve.

Also, to evaluate the fuel economy of a vehicle, it needs to be driven. We use drive traces, which are vehicle velocity profiles, to simulate the driving conditions for the vehicle. We use a closed-loop controller to simulate the driver—this controller manipulates the throttle and brake of the vehicle to track the drive trace, and the fuel consumed by running the simulation over a particular drive trace is used as the objective function that we minimize.

The simulation that we use is of moderate fidelity in that it uses simplified models of combustion and engine flow, and is intended for preliminary control system parameter optimization. We choose to use this simulation model because preliminary parameter optimization with an EA is often an interactive process that requires running the simulation many times, and to do so using a high fidelity model would be prohibitive in terms of CPU time. In order to illustrate what systems and subsystems of the vehicle we simulate, we provide a simplified block diagram of our Simulink software in Figure 1.

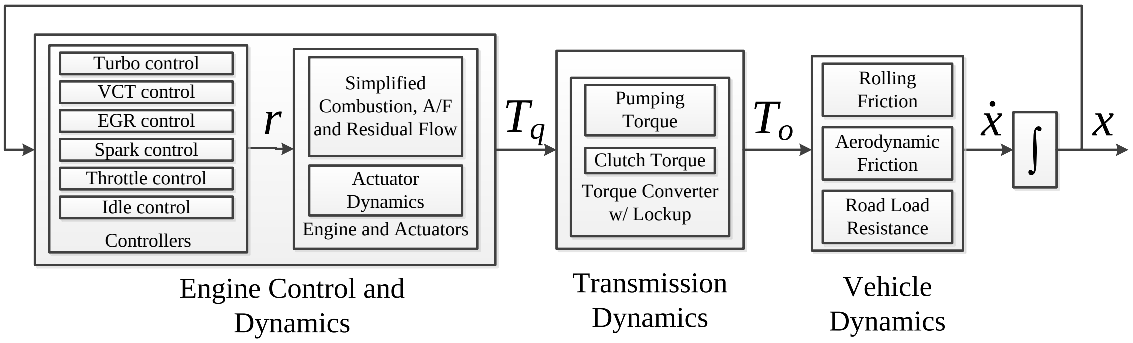

Figure 1: Block diagram of Simulink vehicle simulation, showing subsystems and quantities that are taken into account. *r* is the collection of actuator target (set-point) signals (note that there are closed-loop controllers within the block labeled "Engine and Actuators" that are not depicted) T_q is the torque output of the engine, T_o is the torque supplied to the wheels, *x* is the state of the vehicle, including states of the included subsystems, and $\dot{x}$ is the derivative of the vehicle state.

Following Figure 1 from left to right, we see that the vehicle state (which includes the engine state and other internal variables) is the input to the engine control system. The intake and exhaust camshaft timing actuators in our VCT system are controlled by lookup table mappings that cover a limited domain of engine load and engine speed. The reference values used for controlling the camshaft timing actuators are generated at runtime by linearly interpolating between adjacent table values. The cam timing lookup tables are represented by the "VCT control” block in this diagram. The VCT actuator outputs are modeled in our vehicle simulation as time-delayed and rate-limited versions of the VCT command values from the lookup tables. The “Transmission Dynamics” and “Vehicle Dynamics” blocks represent the lossy delivery of torque from the engine to the wheels, which determines how the vehicle state changes.

We note that it is possible to optimize the output torque of the engine by itself, however, we have chosen to simulate the entire vehicle when evaluating solutions, as fuel economy results in terms of vehicle distance traveled are easier to interpret than fuel economy results in terms of fuel consumed for the engine to meet a torque demand.

2.2 Optimization Variables and Objectives

Our objective is to optimize the fuel economy of a simulated vehicle by adjusting the control of the vehicle’s VCT. Specifically, we make modifications to lookup tables that define the controller set points for the two actuators in this system—the independent variables used for these lookup tables are engine speed and engine load, which can be considered the state variables of the ICE system [29].

We have chosen to modify the tables produced via prior unpublished research. Table I and Table II show the respective intake and exhaust cam timing lookup tables. We have chosen to modify these tables instead of generating completely new tables, because the tables we have chosen to modify produce good results in simulation tests despite being suboptimal. We have chosen this strategy because using BBO to optimize modifications to these tables, rather than using BBO to create new tables, effectively reduces the size of the search space and takes advantage of prior expert knowledge.

	Engine speed (RPM)					
Normalized engine load	700	1000	2000	3000	4000	5000
1.4	-50	-50	-50	-50	-50	-50
1.0	-33	-33	-33	-33	-33	-33
0.8	-33	-30	-33	-25	-33	-33
0.7	-6	-6	11	-33	-33	-6
0.6	11	27	11	0	-33	11
0.5	27	27	11	27	-33	27
0.4	27	27	27	27	-33	27
0.3	27	27	27	27	27	27
0.2	27	27	27	27	27	27

Table I: Intake camshaft timing table, with output values in degrees.

	Engine speed (RPM)					
Normalized engine load	700	1000	2000	3000	4000	5000
1.4	31	31	31	31	31	31
1.0	31	31	31	31	31	31
0.8	24	24	24	24	12	39
0.7	24	5	39	12	39	39
0.6	39	39	39	39	12	39
0.5	39	39	39	39	39	-6
0.4	39	39	39	39	39	24
0.3	39	12	39	39	39	39
0.2	39	-21	39	39	31	31

Table II: Exhaust camshaft timing table, with output values in degrees.

2.3 Radial Basis Functions

In order to use BBO for this problem, we must first cast it as a parameter optimization problem. This means that control solutions must be represented by a set of parameters. We have chosen to use radial basis functions (RBFs) to parameterize our control solutions. RBFs are a class of basis functions, as are sinusoids or complex exponentials in the Fourier series. Specifically, RBFs are functions in one or more dimensions that are symmetric. Examples of RBFs include the Gaussian and the multiquadratic [30].

RBFs are useful for generating arbitrary functions, because any arbitrary function can be produced by a linear combination of an infinite number of orthogonal RBFs. Also, RBFs are especially useful in cases where local changes may need to be made to an arbitrary surface, as changing only one parameter in an RBF-based mapping will produce a localized change in the mapping. An example of an arbitrary function defined by the sum of several RBFs is shown in Figure 2.

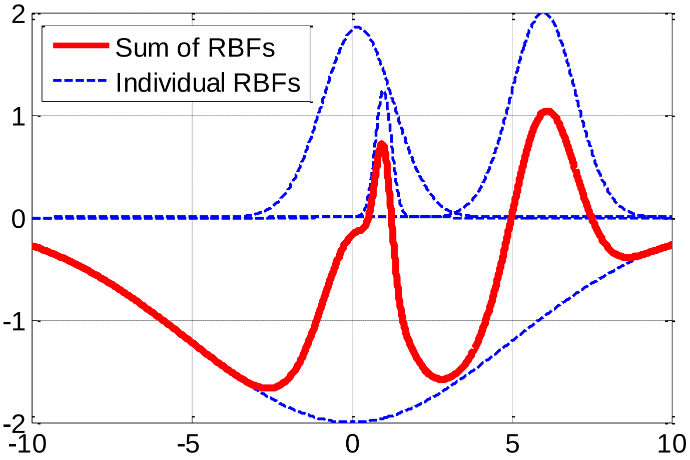

Figure 2: Illustration of several RBFs (indicated by dashed blue lines) and the function produced by summing the RBFs (solid red line.)

2.4 RBF Parameterization Approach

We have chosen to parameterize these modifications (which are arbitrary mappings in (load, speed)-space) with Gaussian RBFs in order to formulate the lookup table optimization problem as a parameter optimization problem for use with BBO. Because the state space of the vehicle is two-dimensional (i.e., the states are engine load and engine speed) we must use two-dimensional Gaussians to map this space. The two dimensional Gaussian is given by the expression

$$f(x,y) = h\exp\left(\frac{1}{2(1-\rho^2)}\left[\frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y} - \left(\frac{(x-\mu_x)}{\sigma_x}\right)^2 - \left(\frac{(y-\mu_y)}{\sigma_y}\right)^2\right]\right) \qquad (2.1)$$

where h is the height or magnitude of the Gaussian, μ_x and μ_y are the coordinates of the center of the Gaussian, σ_x and σ_y are the widths of the Gaussian, and ρ is the correlation between the two dimensions of the Gaussian, and controls how diagonal the Gaussian is in those two dimensions.

We choose Gaussian RBFs because each basis in such an RBF-based mapping contributes locally around a given point to the space spanned by the RBFs while still producing a continuous mapping. As a result, RBF-based mappings are intuitive, since it is easy to visually examine the numerical optimization results. Further, depending on which parameters of the optimization problem are fixed and which are allowed to vary as independent optimization variables, the problem dimension can be made relatively small while still retaining a great deal of flexibility in the variety of solutions that can be represented.

We generate RBF-based lookup-tables by sampling all of our RBFs at all of the (load-speed) points specified by the elements of the original lookup tables, and we add the sampled RBF values to the original tables at these points. Finally, we set any resulting lookup table values outside the range that the actuators can produce, [-27, 50], to the nearest minimum or maximum. We have chosen this approach, where we modify the lookup tables rather than modify the control algorithm, because new lookup tables in the same format as the old ones can immediately replace old ones with minimal engineering effort. We explore different methods of incorporating the RBFs found with BBO to the original lookup tables in Appendix A.

Additionally, we have chosen to fix the centers (μ) and the shapes (the widths, σ and the correlation, ρ) of the RBFs, while optimizing the heights or magnitudes. This reduces the search space of the problem, and thus reduces the search effort necessary to find good BBO solutions. The choices of these fixed parameters is itself an optimization problem. Our approach to this optimization step is to chose these parameters such that the resulting RBFs span the subspace defined by the trajectory of the simulated vehicle through (load, speed)-space given the original actuator lookup tables. Although this optimization step can be formulated in a variety of different ways, including applying clustering methods to the data from this simulation, we have chosen this simple, manual, suboptimal approach to keep our efforts from straying too much from the original problem.

In this method, we chose a fixed number of RBFs to span (load, speed)-space, given a subjective inspection of the number of clusters of data in the simulation trajectory, and we fix the heights of these RBFs to unity. We then manually adjust the locations and shapes of the RBFs until the level curves of the unity-height RBFs visually correlate with the data from the nominal simulation. Figure 3 shows the locations and shapes produced via this manual pre-BBO step using five clusters of (load, speed) data; Figure 4 is similar, but uses 10 clusters of data. Figure 5 and Figure 6 show filled contour plots of the surface made by the sum of the RBFs with their heights set to unity for the cases of five clusters and 10 clusters respectively. These plots are useful for this design process because they qualitatively show us what areas of the controller lookup tables we can adjust with BBO and how well these correlate to the (load, speed) subspace that the vehicle traverses. The centers of the RBFs in the five and 10 cluster formulations are given in captions of Figure 3 and Figure 4. Note that the locations of the centers given in these figures do not directly correspond to the RBF parameters used with BBO. This is because we manually tuned these parameters to better fit the unity height RBF surfaces (as shown in Figure 5 and Figure 6) to the scatter plots of simulation data (as shown in Figure 3 and Figure 4.)

We initially attempted to use *k*-means clustering to find suitable locations for the centers of the RBFs, but could not find a suitable algorithm to subsequently tune the RBF widths after finding the centers. We also tried to use BBO to find the centers, widths, and covariances of the RBFs, using a cost function of the RBF coverage of the state trajectory data points. We found it difficult to find a cost function that encourages RBFs that cover the data points without being too wide. Finally, we note that the expectation-maximization algorithm may be effective for finding RBF parameters, as this algorithm attempts to fit Gaussian distributions to a data set.

We choose to use the five cluster formulation described in Figure 3 and Figure 5, based on results of ad hoc BBO runs. We were able to find better solutions with the five cluster formulation than with the three and 10 cluster formulations we tried, which suggests that five clusters provides a good tradeoff between search space manageability and parameterization flexibility.

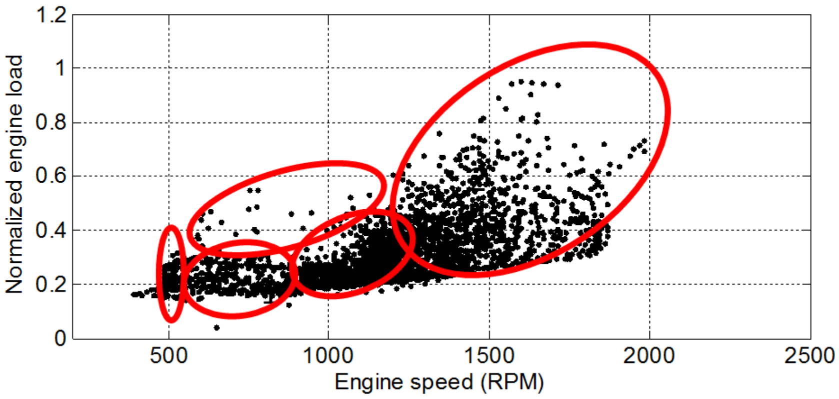

Figure 3: Engine load and speed data scatter plot with circles showing the locations and shapes of five RBFs found with an ad-hoc placement approach. We note that the first RBF is centered at (525, 0.21), the second, third, fourth, and fifth are centered at (700, 0.15), (850, 0.45), (1200, 0.30), and (1600, 0.65). The correlation coefficients for these RBFs are 0.00, 0.15, 0.20, 0.30, 0.45 for the first, second, third, fourth and fifth RBFs respectively.

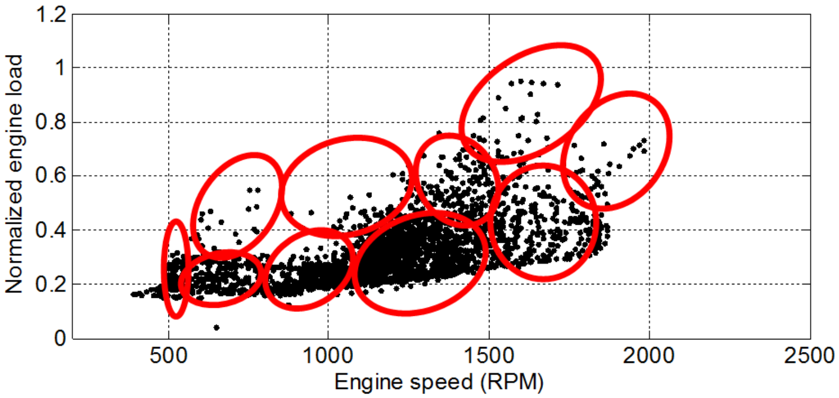

Figure 4: Engine load and speed data scatter plot with circles showing the locations and shapes of 10 RBFs found with an ad-hoc placement approach. We note that the first RBF is centered at (525, 0.21), the remaining RBFs, numbered 2 through 10 are given in the following order, (700, 0.5), (680, 0.22), (920, 0,25), (1050, 0.55), (1400, 0.54), (1300, 0.28), (1610, 0.41), (1600, 0.85), and (1800, 0.70). The correlation coefficients for RBFs 1 through 10 are 0.00, 0.15, 0.70, 0.50, 0.25,
-0.60, 0.30, 0.00, 0.50, 0.65.

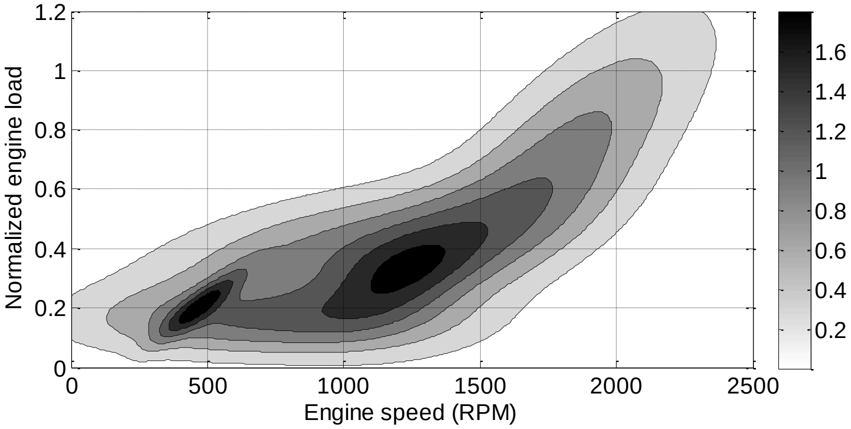

Figure 5: Contour plot of the RBF surface created by setting the RBF heights to unity, in the case of five clusters.

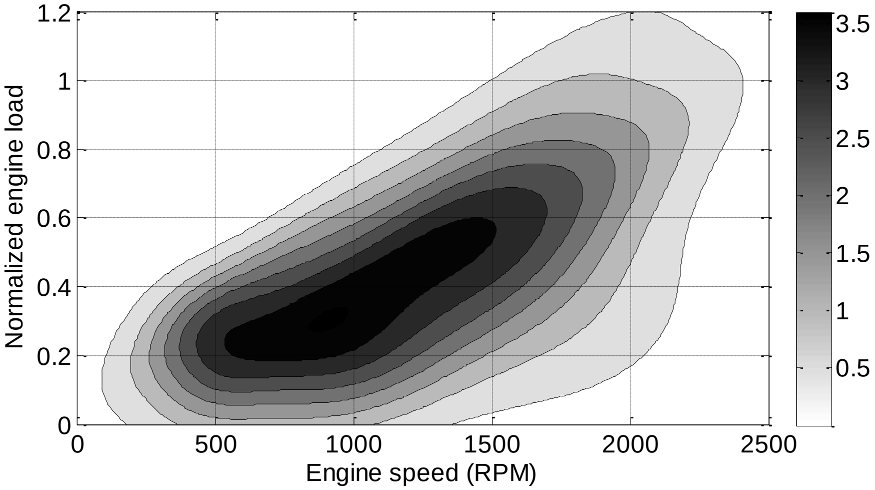

Figure 6: Contour plot of the RBF surface created by setting the RBF heights to unity, in the case of 10 clusters.

Finally, we note we arbitrarily choose to use a search range of [-20, 20] for the RBF heights with BBO. This range was chosen because it is a round number that is roughly one quarter of the full range of camshaft phase angles, [-50, 27], that can be produced by the actuators that we are controlling. Larger search ranges could be used, but the slow dynamics of the actuators require that differences between adjacent camshaft angle values in the lookup tables be small, and giving BBO the freedom to make drastic changes to the tables may encourage BBO to produce solutions that include large changes between adjacent table values. This constraint is important because if the engine control unit (ECU) constantly applies drastic changes to the set points of the camshaft timing actuators (which may occur during slow, stop-and-go traffic), the actuators may not be able to reach their set points. This may reduce the improvements in efficiency that we can obtain with VCT, and may result in an engine that is undesirable to the driver, as it may constantly and abruptly change its performance characteristics. Also, controller stability problems, such as ringing or overshoot may be possible in such a case. We do not attempt to establish a maximum value for the difference between adjacent lookup table values, as our current automotive simulation may not be accurate enough to produce these deleterious effects; instead, we examine our optimization results manually to see if the change between adjacent table values is significantly greater than in the original reference tables. Future work may include increasing the search ranges and constraining the maximum difference between adjacent lookup table values.

2.5 Simulation Drive Traces

Given our chosen parameterization of the lookup table modifications, we can make further adjustments to the problem formulation. We do this by changing the parameters of the simulation so that the cost function of the modified simulation still resembles the original, yet is less expensive to compute. In our case, we can select drive traces for simulation that involve a short simulation time while still containing the driving features that we must optimize the fuel economy over. We have chosen to use the EPA urban dynamometer driving schedule (UDDS), also known as the LA4 cycle [31]. We have also developed a method to pick sub-traces from a given drive trace. Any particular drive trace can be split into a sequence of sub-traces that each begin with a period of idle (vehicle velocity equal to zero) and end with the next period of idle. For instance, the LA4 city trace used in our work can be split into 18 different sub-traces, including one sub-trace that consists only of a period of idling. A plot of the LA4 drive trace with vertical lines denoting the beginning of each sub-trace is shown in Figure 7.

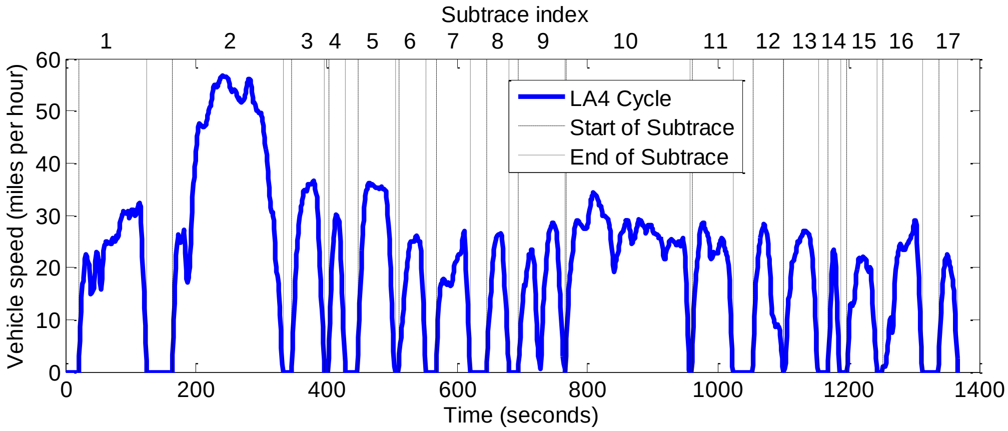

Figure 7: Plot of EPA urban drive cycle with vertical lines showing how we have partitioned it into sub-traces. An index is denoted above each sub-trace, and these are referred to in the remainder of this paper. Not shown in this plot is the 18th sub-trace which consists of all of the idling time between each of the other sub-traces.

The sub-trace approach is a way of extracting sub-functions of the cost function as mentioned in [10]. The details of this approach (especially initial velocity, $v_0 = 0$) are important because they ensure that the initial conditions of each sub-trace are consistent.

We can determine which sub-traces to use for approximating the full cost function trace by examining quantities such as root-mean-square (RMS) velocity and acceleration, the time length of the sub-trace (proportional to the number of samples in the drive trace, since the traces are sampled at a constant rate), and the empirically measured CPU time needed to run the simulation over that sub-trace. Afterwards, we can determine which of these quantities are most strongly correlated with the minimum costs from BBO runs to decide which sub-traces to use. We have examined additional quantities, such as mean velocity and acceleration, but have chosen not to include them in this thesis, because their correlations are almost identical to quantities we include. We exclude these quantities because we are interested in finding the smallest set of independent quantities that captures the nature of a drive trace. Future work may include characterizing the sub-traces by frequency content. It may be desirable to run BBO by simulating over drive traces that include all of the frequency components that a driver would demand from her vehicle, because this would mean that an optimal BBO solution would result in good fuel economy for all of the different speed and acceleration conditions desired by the driver.

2.6 BBO Simulations for Examining Sub-Traces

To execute this test, we ran three Monte Carlo simulations of BBO over each of the 18 sub-traces and evaluated the sub-traces at the four previously mentioned quantities. In order to evaluate how well a sub-trace represents the full LA4 city trace and to provide a basis for comparisons between sub-traces, we evaluate all of the BBO solutions from each run on the full trace during the final generation. In other words, the cost (consumed fuel) values that are discussed in this section are all evaluated on the full city trace, though the candidate solutions were evolved using a particular sub-trace. The BBO parameters for this test are given in Table III. We note that population size was chosen to be a multiple of 12 to reduce overhead, as we run cost function evaluations with 12 MATLAB ® parallel workers on a 24-core AMD Opteron-based™ PC. Also, the number of Monte Carlo simulations was chosen arbitrarily as part of a tradeoff between computational effort and collecting a significant sample size. The worsening probability shown in Table III is the probability that solutions that worsen from one generation to the next replace their parents.

Parameter	Value
Generation count	20
Population size	48
Mutation probability	0.02
Worsening probability	1
Problem dimension	10
Number of elite solutions	2

Table III: BBO parameters for the Monte Carlo simulations where we examine correlations of various quantities with fuel economy for different sub-traces.

Scatter plots of these data with linear curve fitting are shown in Figure 8, Figure 9, Figure 10, and Figure 11. Each point in these scatter plots represents the cost obtained by running the full LA4 cycle over the best solution found in a BBO run over a given sub-trace—the costs shown are averages taken over the three Monte Carlo simulations.

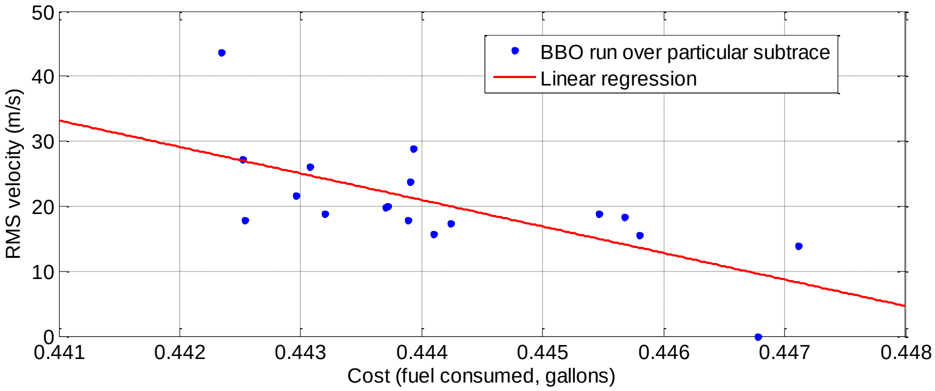

Figure 8: RMS velocity versus the cost obtained by running the best solution of a BBO simulation over the LA4 cycle.

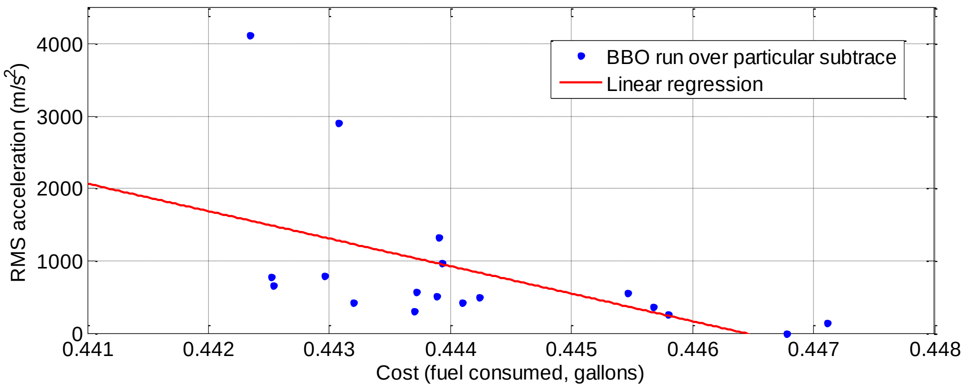

Figure 9: RMS acceleration versus cost obtained by running the best solution of a BBO simulation over the LA4 cycle.

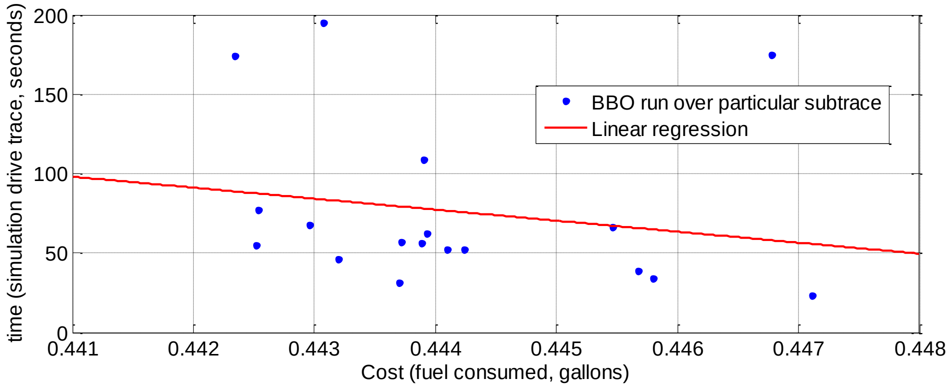

Figure 10: Simulation time length of each sub-trace versus cost obtained by running the best solution of a BBO simulation over the LA4 cycle.

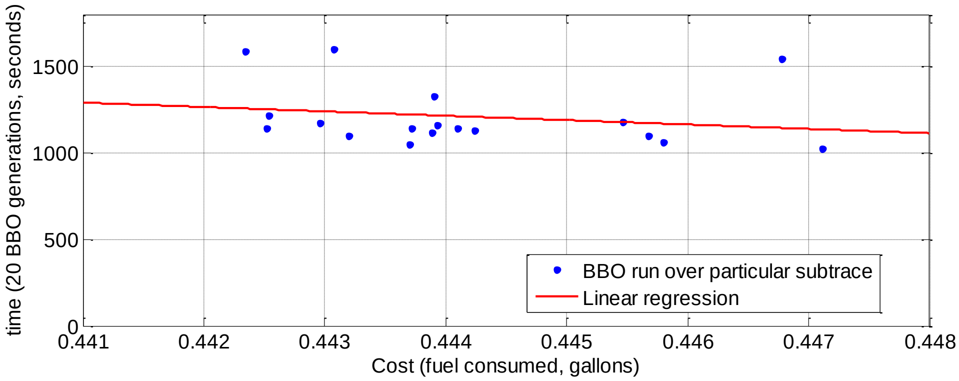

Figure 11: CPU time versus cost obtained by running the best solution of a BBO simulation over the LA4 cycle.

Pearson correlation coefficients between minimum cost and each of the metrics, as well as the associated no-correlation probabilities from these Monte Carlo trials, are shown in Table IV. Specifically, the data in the table are the averages over all sub-traces and over all Monte Carlo simulations (i.e., 18 sub-traces · 3 Monte Carlo trials = 54 simulations). We have chosen to use the Pearson correlation coefficient because straight lines, and thus linear relationships, can be fit to these data most easily.

Metric	Pearson's correlation coefficient:	p-value (no correlation hypothesis)
v_{RMS}	-0.6851	0.0017
a_{RMS}	-0.5299	0.0237
T_{sim}	-0.1907	0.4485
T_{CPU}	-0.1995	0.4273

Table IV: Correlations between minimum cost after a BBO run and the following metrics; RMS velocity, v_{RMS}; RMS acceleration, a_{RMS}; simulation time, T_{sim}; and CPU time, T_{CPU}. Also shown are the associated probabilities that there is no correlation between each metric and final minimum cost.

Given a confidence interval of 5%, the two metrics that have statistically significant correlations are RMS acceleration and especially velocity. Further, the fact that all of the significant correlation coefficients are negative indicates that running BBO over sub-traces that measure higher than average in these metrics is more likely than average to result in better solutions. Since there is little evidence supporting a correlation between simulation or CPU time and minimum cost, we should be able to run for a relatively short simulation time and still get good BBO results, and it doesn't really matter how much CPU time it takes (which may vary depending on how numerically stiff the problem dynamics turn out to be, given the shape of the sub-traces).

Finally, we show the minimum cost after 20 generations of BBO averaged over the Monte Carlo simulations in a bar graph in Figure 12. We include not only the best cost results for the 18 sub-traces, but also the cost for a reference simulation with no changes made to the lookup tables, as well as two particular cases of running BBO over a combination of sub-traces. The first sub-trace combination run consists of the following: for the first 5 generations, the first five sub-traces were run; for generations 6 through 10, the first 10 sub-traces were run; for generations 11 through 15, the first 15 sub-traces were run; and for the rest of the BBO run, the full city trace was run. We refer to this combination as the drive trace "split into quarters." The second sub-trace combination run uses the sum of costs obtained by running over sub-traces 2 and 9. The minimum cost found by optimizing over the entire city trace is also shown for sake of comparison.

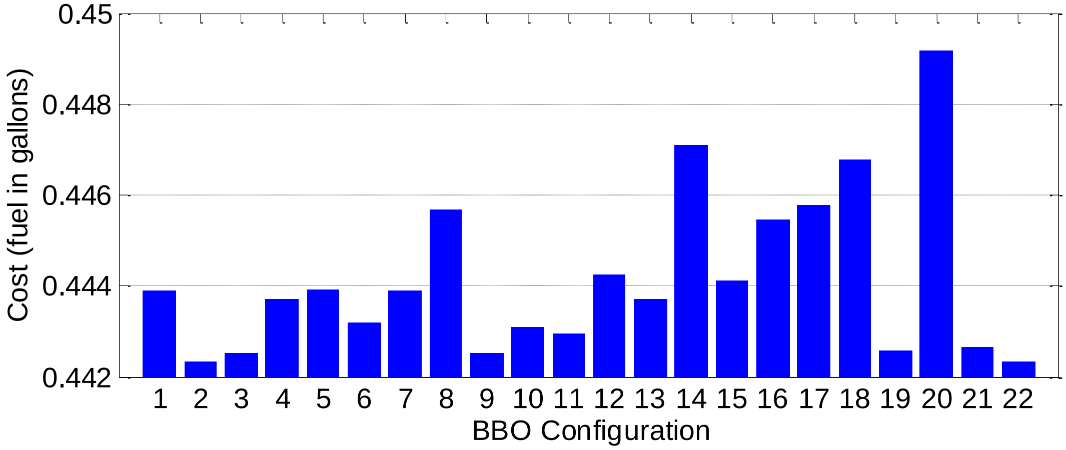

Figure 12: Bar graph showing the average cost of the best solution from BBO for each of the sub-traces. Bars 1 through 18 correspond to the 18 sub-traces, bar 19 corresponds to combination of sub-traces produced by the drive trace split into quarters, and bar 20 corresponds to the cost of the simulation with the original lookup tables. Bar 21 shows a combination of sub-traces 2 and 9, and bar 22 shows the cost obtained from running BBO over the full city trace.

Figure 12 shows us that the two best individual sub-traces to run (i.e., those that resulted in the lowest costs after running BBO) are the second and ninth sub-traces. Further, running BBO with the full city trace results in best costs almost identical to those found by running BBO over the second sub-trace. The combination of the sub-traces 2 and 9 is about as effective as sub-trace 9, which gives poorer costs than sub-trace 2. Further, we see that the combination of sub-traces where the drive trace was split into quarters results in a low best cost on average, however, one that is poorer than running BBO individually on sub-trace 9. It is important to note that these two combinations were chosen arbitrarily, and it is possible that more effective combinations can be found, but running BBO with either combination produces poorer solutions than sub-trace 9 alone.

The conclusion we can draw from these data is that it is best to run over sub-trace 9 than any other individual sub-trace, any combination of sub-traces, or the full drive trace itself. This is because BBO finds better solutions on average when computing cost over sub-trace 9 and the full city trace (i.e., the differences in best costs found with both are insignificant), yet the simulation time of the full city trace is 1371 seconds long, whereas sub-trace 2 is only 174 seconds long. This means that by running sub-trace 2 instead of the full trace, we can reduce simulation time by almost a factor of 8, yet still find the best solutions possible given all of the drive traces that we have evaluated. One thing that should be kept in mind is that these cost data are computed over the full drive trace as a means of comparison; however, this results in a selection bias where the solutions that are considered best are the ones that result in less fuel consumption over the particular driving conditions of the full drive trace, and so we again state that we assume the driving conditions of the city trace represent the average driving conditions faced by the public who would be driving this kind of car.

CHAPTER III

DESIGN AND ANALYSIS OF COMPUTER EXPERIMENTS

3.1 Overview

DACE is a promising Gaussian process surrogate modeling method that applies the algorithm of kriging to the estimation of deterministic computer experiments. DACE has been applied to engineering meta-modeling [15] and reduction of cost function evaluations in EAs [32], and the variety of EA applications of DACE has inspired us to use it for our problem. DACE was first developed by Sacks, Welch, Mitchell, and Wynn as an adaptation of classical design of experiments (DOE) techniques to computer simulation-based engineering design problems [14].

DOE was originally intended for physical experiments that inherently feature random error. In DOE, experiments are proposed to estimate a statistic of a population, and the goal is to find an efficient experiment whose outcome results in the lowest variance in estimation error. However, deterministic computer simulations, such as those used for modeling engineering problems, do not feature random error. DACE makes the false, but useful assumption that the outputs of these computer simulations are actually realizations of stochastic processes.

The assumption given above is false, because, when we program our simulations to generate deterministic outputs, we know that they are deterministic. However, if we treat a deterministic simulation like a black box and evaluate it at several points, all within a given neighborhood, the simulation values within that neighborhood often appear to be normally distributed. Using this tendency, we can estimate simulation values by using sample statistics. This means that DOE techniques can be used for deterministic computer simulations.

3.2 Derivation of DACE

We begin by noting that the equation development in this subsection is adapted from [33], with changes as seen appropriate. In DACE, we make the assumption that we can use information from M cost function evaluations, denoted $f(x) = [f(x_1) \dots f(x_M)]$, to approximate the cost function using the following expression

$$f(x_i) = \mu + \varepsilon(x_i)\ ;\ i \in [1 \dots M] \tag{3.1}$$

where x is a vector of M candidate solutions $\{x_i\}$ (each of which are n-dimensional vectors, where n is the optimization problem dimension), μ is a scalar constant, and ε is a scalar-valued, Gaussian random function, with a mean of zero, and a standard deviation of σ^2.

Note that $f(x)$ is a random vector with a multivariate Gaussian distribution. This means that its probability density function is given by

$$\mathrm{PDF}\big(f(x)\big) = \frac{1}{2\pi^{M/2}|R|^{1/2}}\ \exp\left(-\frac{(f(x)-\mu 1_M)^{\mathrm{T}}C^{-1}(f(x)-\mu 1_M)}{2}\right) \tag{3.2}$$

where C is the matrix of covariances between each element of $f(x)$, and 1_M is an M-element column vector composed of all ones. Also, note that each element of $f(x)$ has the same variance, and so the covariance and correlation matrices of $f(x)$ are related by $C = \sigma^2 R$. This means that we can rewrite Equation 3.2 as

$$\mathrm{PDF}\big(f(x)\big) = \frac{1}{2\pi^{M/2}\sigma^{M}|R|^{1/2}} \exp\left(-\frac{(f(x)-\mu 1_M)^{\mathrm{T}} R^{-1}(f(x)-\mu 1_M)}{2\sigma^2}\right) \tag{3.3}$$

where *R* is the correlation matrix, given by

$$R = \begin{bmatrix} \rho_{1,1} & \cdots & \rho_{1,M} \\ \vdots & \ddots & \vdots \\ \rho_{M,1} & \cdots & \rho_{M,M} \end{bmatrix} \tag{3.4}$$

DACE assumes that different values of *x* must be correlated in some way. After making this assumption, all we need to do to fit DACE models to our data is to pick a correlation function, and then find correlation function parameters that result in the best fit. The correlation function used most often in the literature is a function of a weighted distance metric—this distance metric and the correlation function are given by

$$d_{i,j} = \sum_{k=1}^{n} \theta_k \left|x_i(k) - x_j(k)\right|^{p_k} ;\ k \in [1 \dots n] \tag{3.5}$$

$$\rho_{i,j} = \mathrm{Corr}\left(f(x_i), f\left(x_j\right)\right) = \exp\left(-d_{i,j}\right) \tag{3.6}$$

In this case, $\mu \in (-\infty, \infty)$, $\sigma^2 \in [0, \infty)$, $\theta_k \in [1, 2]$, and $p_k \in [0, \infty)$ (where $k \in [1\dots n]$) are the model parameters that we must fit to the data. This means that we have a total of 2+2*k* model parameters.

The distance metric from Equation 3.5 is chosen such that candidate solutions, x_i and x_j that are near each other have high correlations (approaching 1), and candidate solutions that are far apart have low correlations (approaching 0).

The method for fitting the parameters of a statistical model or predictor to some sample data is called maximum-likelihood estimation. By definition, the likelihood function of these model parameters given *f*(*x*), is the PDF of *f*(*x*) given the model parameters. This means that the expression for PDF(*f*(*x*)) from Equation 3.3 is the likelihood function. We can then find closed form expressions for μ and σ^2 that maximize the likelihood function. These are

$$\hat{\mu} = \frac{1_M{}^{\mathrm{T}}(R^{-1})f(x)}{1_M{}^{\mathrm{T}}(R^{-1})1_M} \tag{3.7}$$

$$\hat{\sigma}^2 = \frac{(f(x) - \mu 1_M)^{\mathrm{T}} R^{-1} (f(x) - \mu 1_M)}{n} \tag{3.8}$$

We can then substitute the maximum likelihood estimates for μ and σ^2 into the likelihood function, which gives us a likelihood function that has only $2k$ model parameters to fit. We use the fmincon function provided in MATLAB to find θ and p that maximize this so-called concentrated likelihood function. The fmincon function implements constrained minimization for general nonlinear problems. In order to cast the likelihood maximization problem as a minimization problem, we minimize the reciprocal of the likelihood function, noting that several of the factors in this equation are not functions of θ and p, and thus can be regarded as constants during optimization. This yields the following cost function for minimization

$$J_{DACE} = \sigma^M |R|^{1/2} \tag{3.9}$$

Finally, once we have found optimal θ and p values, we can use our DACE model as a predictor to estimate the cost function at a new point x^* with

$$f(x^*) = \mu + r(x^*)^{\mathrm{T}} R^{-1} (f(x) - \mu 1_M) \tag{3.10}$$

where r is the vector of correlations between x^* and each of the M sample points in the vector x.

It should be noted that the determinant of the correlation matrix, R, is used in Equation 3.9 for the model fitting process, and its inverse is taken in Equation 3.10 when using the model to estimate the cost function. If R becomes ill conditioned, then we will have difficulties both fitting and evaluating DACE models. We have taken several steps to keep this from happening, and we describe them in detail in Appendix B.

CHAPTER IV

BBO SOFTWARE FOR CAM TIMING OPTIMIZATION

4.1 Overview of Cam Timing BBO Software

We describe all of the problem specific modifications we have made, and the significance of each, in this chapter. We begin by describing our approach of running different simulation drive speed traces during a BBO run, then we provide a brief discussion of the methods of parallelizing the cost function evaluations, then we talk about keeping a record structure of all of the evaluated BBO solutions, then we discuss our approach for applying DACE to BBO, and finally we look at the entire software architecture at a high level and provide some flow charts for illustration.

4.2 Varying Simulation Drive Traces in BBO

We have implemented a feature allowing any number of different simulation drive traces to be run in different generations in our BBO software. For instance, we can run both sub-traces 2 and 9 for the first three generations of BBO, then the full city trace for generations 4 and 5. We have accomplished this by creating a lookup table whose rows contain all of the sub-trace indices we wish to run in a given generation. The table that is generated for the hypothetical case we describe in this paragraph is shown in Table V.

Generation Index	First Drive Trace	Second Drive Trace
1	Sub-trace 2	Sub-trace 9
2	Sub-trace 2	Sub-trace 9
3	Sub-trace 2	Sub-trace 9
4	City trace	N/A
5	City trace	N/A

Table V: Example of a lookup table used to select which drive traces are simulated each generation.

This software feature allows us to implement schemes where we begin by running BBO over a small number of sub-traces to reduce computational effort, yet still get the BBO population to start converging to the neighborhood of the best solutions. Then, as the BBO run progresses, we start adding more subtraces so that BBO converges to solutions that more closely reflect the minima of the cost function defined by the full city trace. We have found that there is significant CPU time overhead involved in running many different subtraces each generation, because the initialization step in the vehicle simulation takes a significant amount of time, and running over multiple subtraces does not yield the improvement in the quality of the best solutions to warrant doing this.

4.3 Parallelized Cost Evaluation

In order to leverage the increasing number of processor cores in modern computers to reduce the computation effort of a BBO run, we have chosen to use the "spmd" program control method from the MATLAB Parallel Computing Toolbox ®. The spmd function creates copies of the variables in a MATLAB workspace, and runs the same program using several parallel workers (or jobs) over each copy. The output data from these parallel workers can then be merged by the user after all of the workers finish executing the program.

In order to parallelize the cost function evaluation step with spmd, we split all of the candidate solutions that we wish to evaluate into *n* number of groups, corresponding to the number of parallel workers chosen (which cannot be greater than the number of CPU cores). Inside each of the parallel spmd programs, we have a for loop that computes the cost of all of the individuals within a given group in series. We illustrate how cost function parallelization is used within our BBO software later in Section 4.6.

4.4 Recording and Using Costs of Previously Evaluated Solutions

We make the following modifications to standard BBO. First, we add a structure to the BBO software that contains every solution that we evaluate over the course of a run, along with its associated cost; we call this structure the record. At the end of every generation, we add all of the solutions whose cost has been calculated during the most recent generation into the record, making sure not to add duplicate entries. When computing costs after applying the evolutionary operators, we can check if BBO solutions exist within the record, and if they do, we do not need to compute their costs again; instead, we can look them up in the record.

We make sure to handle the case where the cost function for BBO may change from one generation to the other. As we have mentioned, we have the capability to change the number or indices of the drive traces we can simulate over, from generation to generation. If we change the simulation drive traces, then we necessarily change the cost function, and the costs of solutions that we have recorded are no longer valid for the new cost function. Because of this, we delete the solution record and start over at the beginning of BBO generations where we change the cost function. We illustrate our use of solution recordkeeping later in Section 4.6.

Further, the record can be used as a pool from which to draw samples for generating DACE models. It is useful to have more points available to sample from, because more points means we have more flexibility in choosing a sample set for modeling. More information on our application of DACE to BBO can be found in the following subsection.

4.5 DACE Application to BBO

Because DACE has been applied to other EAs before, it makes sense that it can be applied to BBO as well. Although there are several evolution control strategies (i.e., several algorithms for deciding when to estimate cost with DACE instead of calculating it) [32], we have chosen to develop our own strategy for using DACE with BBO to leverage the record of evaluated candidate solutions that we keep as a pool of samples for generating DACE models.

During each generation, we determine whether to estimate or calculate the cost of a candidate solution by comparing the Euclidean distance between the solution and the closest DACE sample point to it in search parameter space. Two strategies for making this decision include picking a fixed number of candidate solutions to estimate and choosing those who are closest to their nearest sample point, and estimating only those solutions that are closer than a given distance threshold to their nearest sample points. Figure 13 illustrates the concept of the closest distance from a point that we are estimating and the nearest DACE sample to it, in a hypothetical, one-dimensional case. The distance threshold strategy makes sense because the mean-squared error (MSE) of a DACE model is zero at the sample points used to fit the model, and generally increases as one looks further and further away from the sample points [33]. The main drawback to this method is the challenge associated with choosing the distance threshold given how aggressively we want the EA to estimate the cost function. We can also use the MSE expression from [33] instead of distance to decide which solutions to estimate.

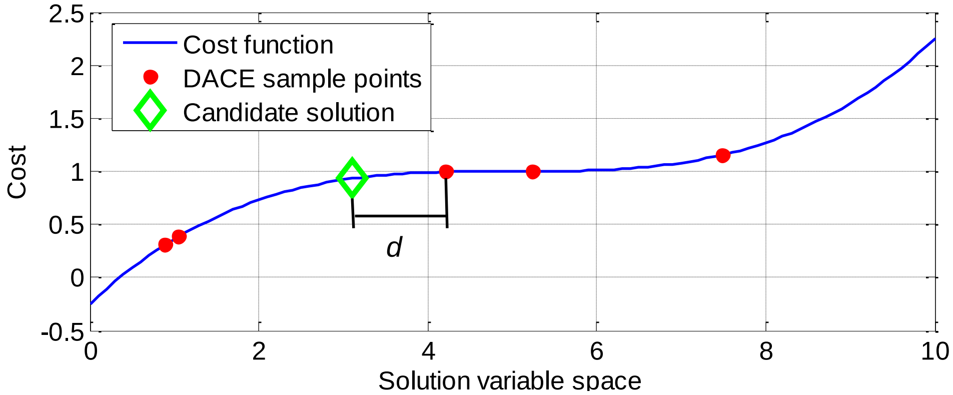

Figure 13: Drawing depicting the distance, *d*, between a solution to be estimated and its closest DACE sample.

The set of samples chosen to generate a DACE model also strongly affects the performance of an EA using DACE. Latin hypercube sampling is often used for fitting DACE models [33], because a Latin hypercube sample set can better represent a distribution than a uniform sample set, given certain conditions [34]. In contrast, building a DACE model from normally distributed samples is a bad idea, because the samples will tend to be close together, and the correlation matrix, R, which must be inverted in the process of fitting or sampling from a DACE model, becomes nearly singular [33].

In the case of using DACE to estimate an expensive cost function in an EA, it is undesirable to compute all of the cost function evaluations necessary to achieve a Latin hypercube sampling; instead, we want to use the solutions from the EA population that have already been evaluated. This poses a numerical problem, because the solutions in an EA population tend to cluster around local optima during an EA run, and thus the R matrix developed while fitting DACE will be ill-conditioned. For this reason, it may be useful to sample from a record of all of the solutions that have been evaluated over the course of a run in such a way that the resulting sample set will have desirable properties. One important property is that the sample set should result in a well conditioned R.

We have developed a simple, sub-optimal sampling heuristic in which solutions from the record obtained using the method in the previous paragraph are selected, one-by-one, such that the resulting sample set is well spread out. The first sample is chosen as the solution closest to the mean of the record solutions, and the subsequent samples are chosen as those furthest away from the mean of all of the previously chosen samples. Future work may include a more optimal approach; perhaps involving finding a subset of the solution record that most closely correlates with a desired distribution. This simple heuristic algorithm was chosen because it provides good results, yet does not distract us from the underlying ICE optimization problem. Algorithm 2 provides a pseudocode description of this "spread out" sampling heuristic and Figure 14 illustrates a particular hypothetical, two-dimensional case.

```
$i_{min} = \underset{i=[1,N]}{\operatorname{argmin}} \|\bar{x} - x_i\|$
$y_1 = x_{i_{min}}$
$g_1 = i_{min}$

for $j = 2$ to $M$
    $k_{max} = \underset{k=[1,N]\notin g}{\operatorname{argmax}} \|\bar{y} - x_k\|$
    $y_j = x_{k_{max}}$
    $g_j = k_{max}$
end
```

Algorithm 2: Pseudocode representation of our DACE sampling heuristic, where *N* is the size of the solution record, *M* is the desired number of samples, *x* is the vector of solution vectors in the record, $\bar{x}$ is the mean of the recorded solutions, *y* is the resulting set of DACE samples, *g* is the set of indices corresponding to previously chosen record solutions and is used to implement sampling without replacement, and i_{min} and k_{max} are the indices that optimize their preceding equations.

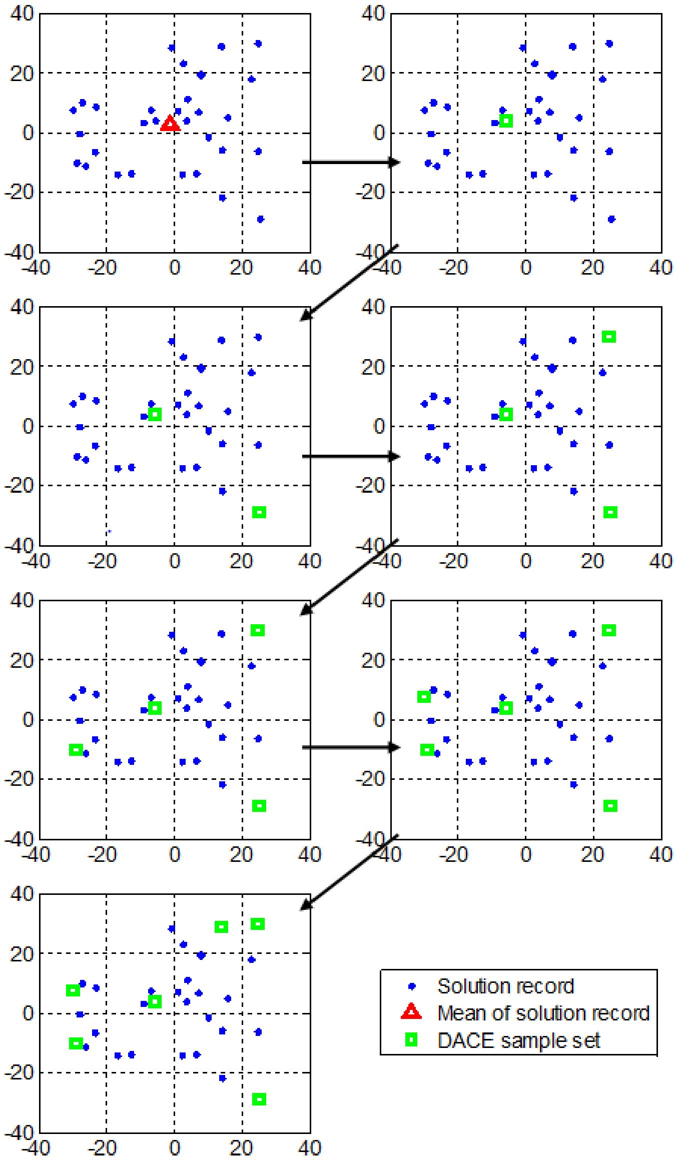

Figure 14: Illustration of a test case of our DACE sampling heuristic with six samples drawn from a set of 30 samples taken from a uniform distribution.

It is also important to note that a DACE model can be used as an additional source for generating candidate solutions. If a DACE model is particularly accurate around the global optimum, we can use the optimum of the DACE model to estimate the global optimum of the search space. In a BBO algorithm, we can use DACE as an evolutionary operator to replace some of the poorest solutions with solutions that optimize a DACE model. Alternatively, we can replace poor BBO solutions with solutions at points in the search domain where the DACE model has high error—after we evaluate the cost of these solutions, we can use them as samples for generating subsequent DACE models, and thus reduce the DACE model error within the vicinity of these new samples. We are motivated to consider this, because optimizing a DACE model is generally much easier than directly optimizing the cost function that the model represents. We have not pursued this idea any further, but it remains as a possibility for future work.

4.6 BBO Software Flow Charts

Figure 15 shows a flow chart that illustrates the program flow of our BBO algorithm. The details of the cost function evaluation step in Figure 15 are illustrated in greater detail in Figure 16. As can be seen from these figures, the step for evaluating the costs of the BBO population has several substeps. Most of the improvements to CPU time are made in the cost evaluation step.

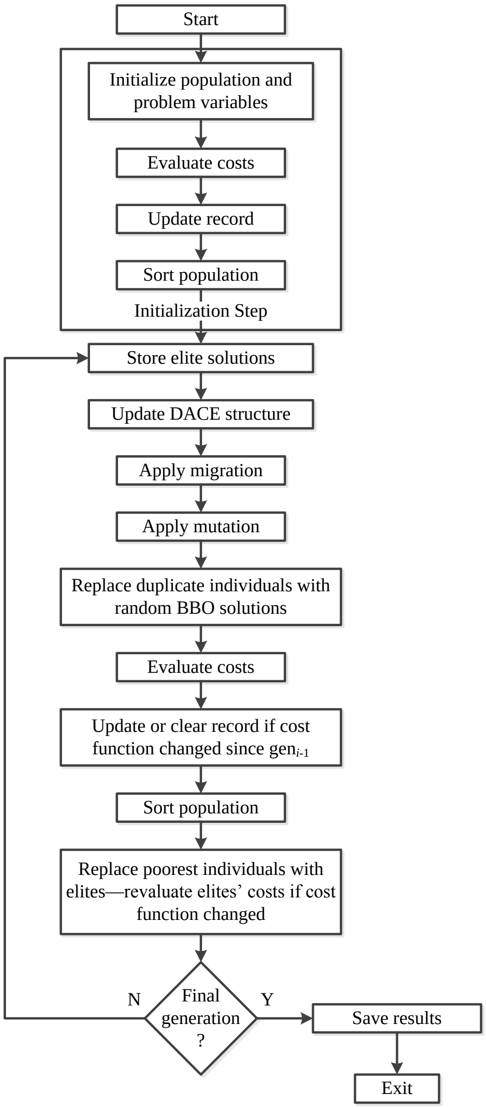
Start
Initialize population and problem variables
Evaluate costs
Update record
Sort population
Initialization Step
Store elite solutions
Update DACE structure
Apply migration
Apply mutation
Replace duplicate individuals with random BBO solutions
Evaluate costs
Update or clear record if cost function changed since gen$_{i-1}$
Sort population
Replace poorest individuals with elites—revaluate elites' costs if cost function changed
N
Final generation ?
Y
Save results
Exit

Figure 15: Flow chart illustrating the program flow in our BBO software. Note that the "Evaluate costs" block shown twice in this figure is expanded in Figure 16.

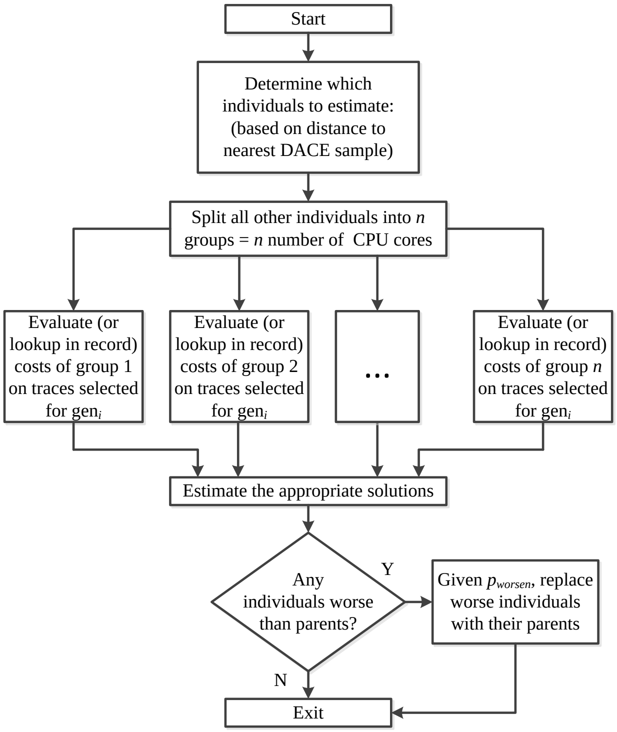
Start
Determine which individuals to estimate: (based on distance to nearest DACE sample)
Split all other individuals into n groups = n number of CPU cores
Evaluate (or lookup in record) costs of group 1 on traces selected for gen$_i$
Evaluate (or lookup in record) costs of group 2 on traces selected for gen$_i$
...
Evaluate (or lookup in record) costs of group n on traces selected for gen$_i$
Estimate the appropriate solutions
Any individuals worse than parents?
Y
Given p_{worsen}, replace worse individuals with their parents
N
Exit

Figure 16: Flow chart indicating how the record of previously evaluated solutions, worsening rate, variability of choices of simulation drive trace, estimation of individuals with DACE, and cost function parallelization are used within our BBO software.

CHAPTER V

OPTIMIZATION PARAMETER STUDIES

5.1 Optimization Parameter Study Motivation

It is important to examine the characteristics of the optimization method chosen for any particular problem, since one algorithm or one set of optimization algorithm parameters may be better suited for a given problem than others. This is a consequence of the No Free Lunch theorem, which states that all algorithms perform equally well on average when tested on the most general class of problems [12]. This motivates us to study the DACE and BBO parameters that we use for our variable camshaft timing problem, since an algorithm that takes advantage of problem specific knowledge usually performs better on that particular problem than algorithms that do not.

While we could optimize these parameters, the number of the parameters that could be varied is large, and the problem of optimization algorithm parameter optimization, when taken to its logical conclusion, becomes one of infinitely many nested optimization problems (i.e., we use one algorithm to optimize another, which optimizes another, ad infinitum).

The Vertical No Free Lunch Theorem given in [35] states that the difficulty of a search increases exponentially as more and more nested levels of search are added, unless some information about the higher level search spaces is known. This means that a search for optimal EA parameters is only useful if some information is already known about the search space for these parameters. As EA researchers, we know what ranges of EA parameters work well for problems in general, and so we can incorporate this information into a meta search that may yield a better EA for a given problem.

For this reason, we consider it sufficient to evaluate our version of BBO throughout the optimization parameter search domain, varying one dimension at a time while holding the rest constant. In doing so, we can estimate the relationships between each of the algorithm's independent variables, which allows us to find suboptimal but effective search parameters.

5.2 BBO Parameter Studies

In order to find effective BBO parameters for our particular problem, we first define a reference set of BBO variables which are shown in Table VI and which represent typical parameters used in the literature. We then vary parameters such as population size, mutation rate, and worsening rate, one at a time. We note that we ran these tests over only sub-trace 10 to facilitate the Monte Carlo approach, since running a BBO simulation over even a single sub-trace can take more than 3 hours depending on the BBO parameters.

Parameter	Value
Generation count	50
Population size	100
Mutation probability	0.02
Worsening probability	1
Problem dimension	10
Number of elite solutions	2
Simulation drive trace	sub-trace 10

Table VI: Nominal BBO parameters for BBO parameter studies.

5.2.1 Population Size

First, we examined population size by running eight Monte Carlo simulations with population size equal to the values: 25, 50, 75, 100, 150, 200, and 300.The cost of the best solution from each Monte Carlo simulation is represented by a point on a scatter plot in Figure 17.

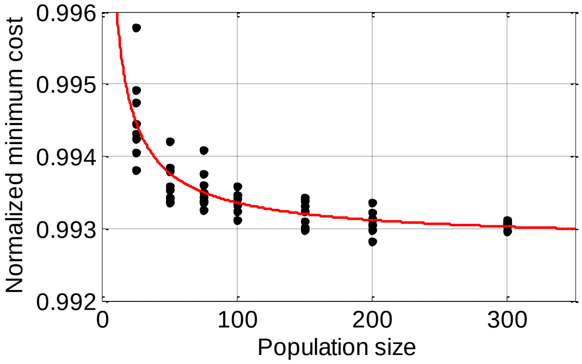

Figure 17: Scatter plot of normalized minimum cost after 50 BBO generations versus population size. A power function curve is also shown.

Because this relationship appears to be a monotonic one (as one would expect), there is no optimum and so we must find a suitable tradeoff instead. It appears that we reach a point of diminishing returns after population size increases past 200, so this may be a useful value for population size, especially considering that we can estimate the cost of many candidate solutions with a DACE model. This implies that up to 200 parallel processors would be an attractive setup for a BBO run, but any more than that would be beyond the point of diminishing returns.

In order to quantitatively gauge the relationship between population size and the cost of the best solution after a BBO run, we compute the Spearman rank-based correlation coefficient, ρ. Spearman's correlation coefficient is useful for this case, because we are interested in gauging the strength of a general monotonic relationship between the two dimensions, not necessarily a linear one. For this data, ρ = -0.8709 and the probability of the no-correlation hypothesis is $2.7160 \cdot 10^{-18}$.

5.2.2 Mutation Probability

Next, we examine mutation probability. We started with the reference BBO parameters from Table VI and ran eight Monte Carlo simulations for each of the following mutation probability values: 0.01, 0.05, 0.10, and 0.20. The reference BBO runs are also included in these data, for which mutation probability was set to 0.02. We show a scatter plot of these data, including a spline curve fit to the data in Figure 18. A spline, quadratic, or any other concave up curve that we fit to this data will suggest that the minimum is in the neighborhood of mutation probability of 0.05. Given that the rest of the BBO parameters are set to the reference values from Table VI, this data suggests that mutation probability around 0.05 is optimal. This data appears to suggest a non-monotone relationship between minimum cost and mutation probability (one with a global minimum), and so we do not attempt to find a correlation coefficient.

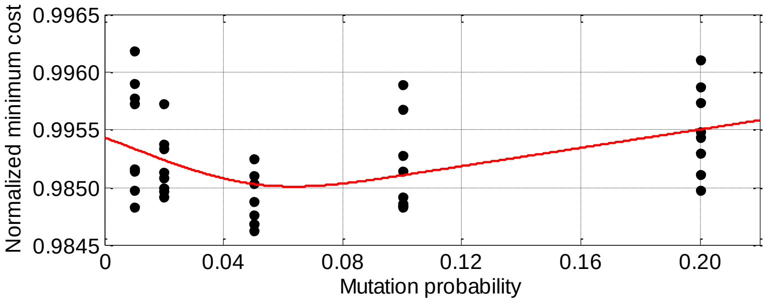

Figure 18: Scatter plot of normalized minimum cost after 50 BBO generations versus mutation probability. A spline curve fit to this data is also shown.

5.2.3 Worsening Probability

Next we look at results of eight Monte Carlo simulations where we set the worsening probability to the following values: 0.00, 0.25, 0.50, and 0.75. Again, we include the results of the reference BBO runs, where worsening probability was set to 1.00. These data are plotted in Figure 19 with a straight line fit to the data.

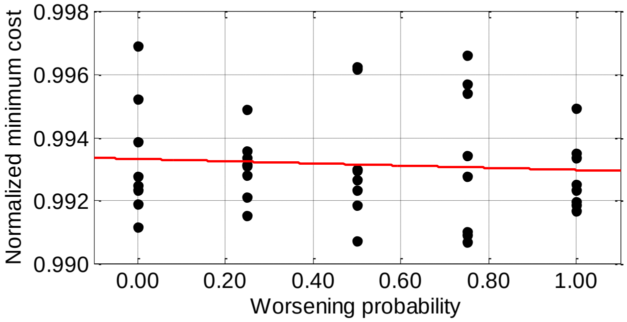

Figure 19: Scatter plot of normalized minimum cost after 50 BBO generations versus worsening probability. A straight line curve is also shown. Cam optimization results

The Pearson correlation coefficient for this data was computed to be ρ = -0.0747, and the probability of the no-correlation hypothesis is 0.6467. These values indicate that a correlation between minimum cost and worsening probability is unlikely given the nature of our problem and the values of the BBO parameters that we held constant. Thus, we can say that worsening probability does not matter much in our case. This is an interesting result, because one may expect a worsening probability of 0 to contribute to stagnation of the BBO population, because if solutions that perform worse after being modified by the migration and mutation operators are never used, and instead are kept as they were before being affected by these operators, the BBO population will likely change much less from generation to generation, however, it seems that sufficient evolution occurs in all cases of worsening rate to produce comparably good solutions. We leave more rigorous examination of worsening probability for future work. Finally, we note that we choose to fit a straight line and use the Pearson correlation coefficient for these data, because a linear relationship for this data appears as plausible as any other kind of relationship.

5.2.4 Number of Generations

We also examine the convergence of minimum cost in BBO as a function of the number of generations. We ran 10 Monte Carlo simulations for 200 generations each with the parameters from Table VI held constant. Figure 20 shows the average of the best costs from each Monte Carlo simulation with standard deviation bars.

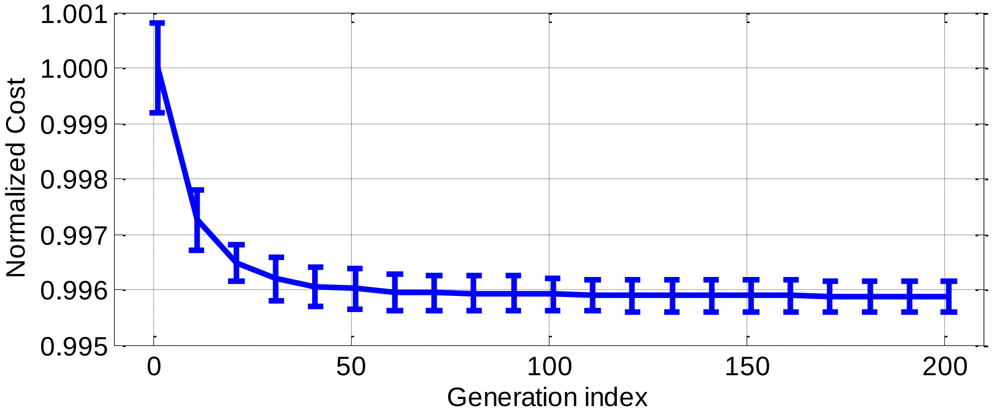

Figure 20: Normalized minimum cost averaged over the 10 Monte Carlo simulations with vertical bars drawn to indicate minimum cost standard deviation taken over the 10 Monte Carlo simulations.

One way of quantifying the generation where the BBO population converges to a minimum solution on average, is to compare the sample sets of the minimum cost data from the final generation with the data from the previous generations. The first generation (that is, the one with the lowest index) whose cost data come from a distribution statistically similar to the final generation's data can be considered the generation at which BBO converges. Using the *t*-test with a confidence interval of 0.05, the first generation to be statistically similar to the final one is generation 40. Figure 20 corroborates this conclusion, however, the use of different confidence intervals will suggest different generations of convergence. In conclusion, this data tells us that we can run for 40 generations and get minimum cost solutions that are not significantly different from those that we would get from running for 200 generations. This also suggests that our choice of 50 generations for the nominal BBO parameters given in Table VI is well justified.

5.2.5 Convergence Properties

Another thing worth examining is the effect that variations in parameters like population size or mutation probability have on the convergence properties of BBO. We take advantage of the eight Monte Carlo runs presented earlier in this section to examine the minimum cost convergence as we vary these parameters. Figure 21 shows the normalized minimum cost for each population size that we evaluated, versus generation index, averaged over all 8 Monte Carlo simulations. Note that the last generation is not shown in this figure, since this generation is evaluated over the full city trace for purposes of comparison in Figure 17, Figure 18, and Figure 19, and would be impossible to plot on the vertical scale used in this figure. Further, standard deviation bars are not shown for this data, since it is difficult to draw them legibly with the plots for each population size value—we consider it sufficient to include values for the standard deviation of minimum costs averaged over all of the generations of the run, computed over all Monte Carlo simulations, since the standard deviation does not change much from generation to generation. These values are given in Table VII.

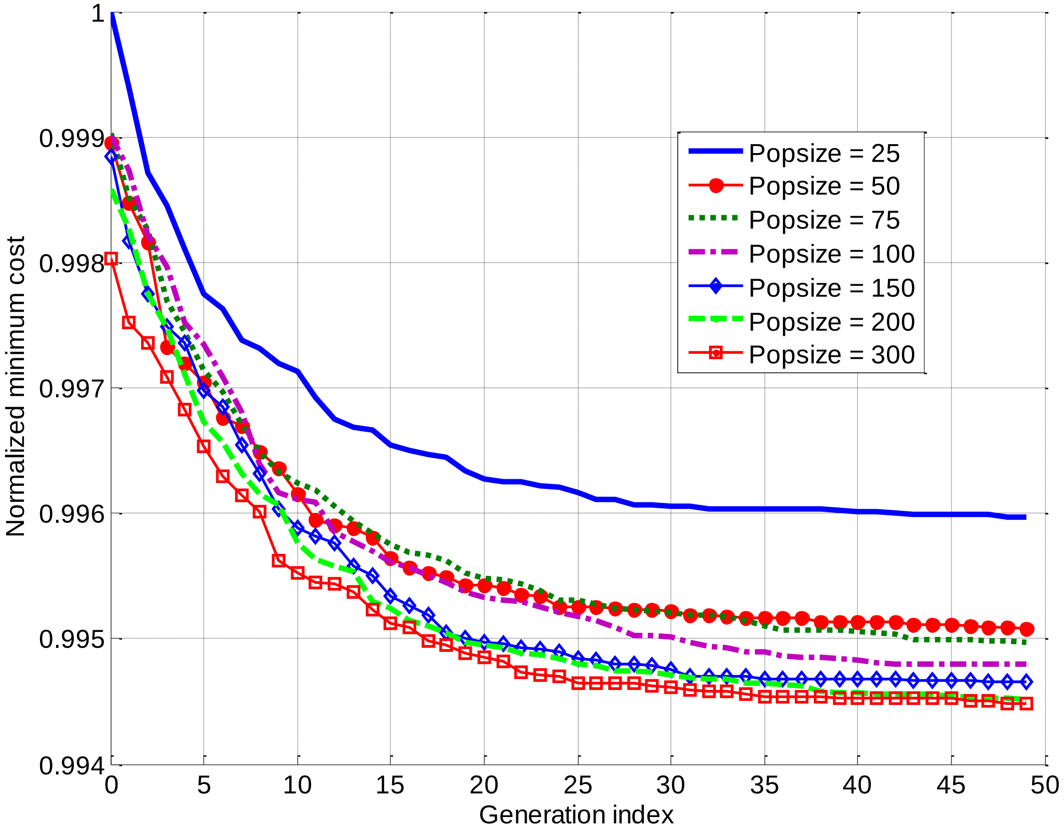

Figure 21: Family of plots in population size showing normalized minimum cost, averaged over 8 Monte Carlo trials as a function of generation index.

Population Size	Average Standard Deviation
25	1.26E-05
50	7.19E-06
75	6.61E-06
100	4.75E-06
150	5.13E-06
200	4.66E-06
300	3.51E-06

Table VII: Standard deviation of minimum cost for each population size evaluated, averaged over all generations (not including the final generation where cost was computed over the full city trace). Note that these data are not normalized as are the data included in the previous figure.

As we can see from Figure 21, the minimum cost vs generation curves seem to converge just as quickly as the population size is varied; the curves differ mostly by a vertical shift. Further, the standard deviation of minimum costs goes down as population size goes up. This is expected, since a small population necessarily covers a smaller proportion of the search space, and thus one would expect more variability in comparing minimum costs of smaller populations on average. These facts mean that population size does not significantly affect convergence speed of minimum cost solutions in BBO when applied to our problem.

Next, we observe the effect of differing mutation rate on BBO convergence speed. Figure 22 shows the normalized minimum cost versus generation index as mutation probability is manipulated from 0.01 to 0.20, and Table VIII shows the standard deviations for these mutation rates.

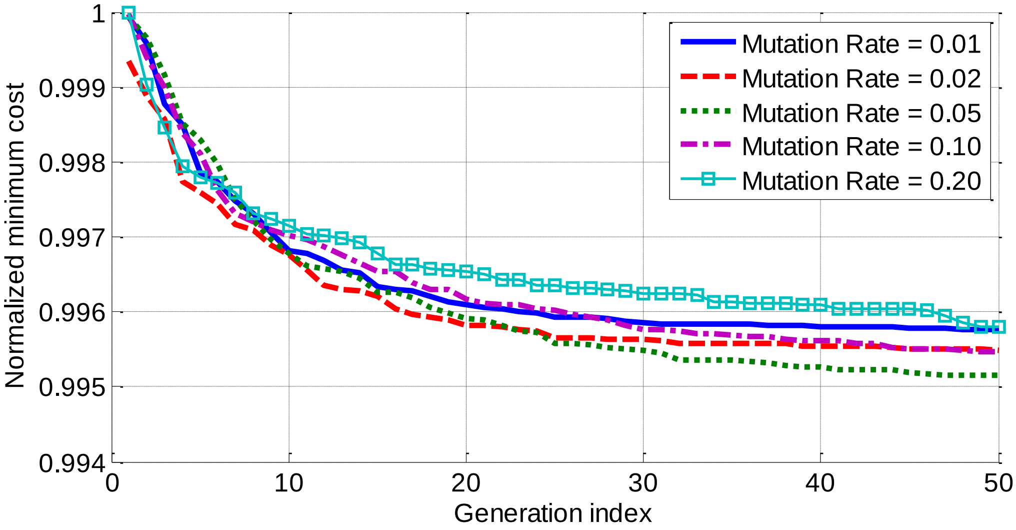

Figure 22: Family of plots in mutation rate showing normalized minimum cost, averaged over 8 Monte Carlo trials as a function of generation index.

Mutation Probability	Average Standard Deviation
0.01	1.14E-05
0.02	7.19E-06
0.05	9.77E-06
0.10	8.76E-06
0.20	7.48E-06

Table VIII: Standard deviation of minimum cost for all of the mutation rates evaluated, averaged over all generations (not including the final generation where cost was computed over the full city trace). Note that these data are not normalized as are the data included in the previous figure.

The plots for mutation rate seem to differ by a horizontal scaling instead of a vertical shift, which suggests that if mutation probability has any significant affect on minimum cost, it is likely to have a significant effect on the convergence speed rather than any other kind of effect. Also, the standard deviation data does not appear to show a trend, and examination of the scatter plot in Figure 17 suggests that mutation probability does not have a significant effect on the variability of best costs obtained after running BBO. This, and the fact that the plots in Figure 22 are closely clustered together, suggests that mutation rate does not have a significant effect on convergence or minimum cost, though analysis with a larger sample size of Monte Carlo simulations may suggest a different conclusion.

We also examine the effect of worsening rate on convergence speed. Figure 23 shows the minimum cost versus generation for the worsening rate simulations, and Table IX shows the standard deviations for these data.

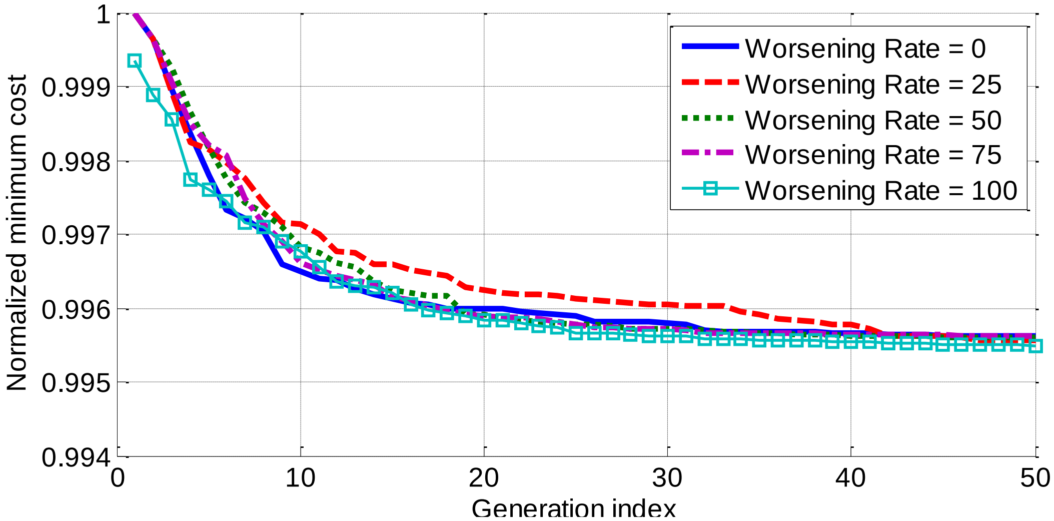

Figure 23: Family of plots in worsening rate (percentage) showing normalized minimum cost, averaged over 8 Monte Carlo trials as a function of generation index.

Worsening Probablity	Average Standard Deviation
0.00	1.01E-05
0.25	1.11E-05
0.50	1.19E-05
0.75	1.29E-05
1.00	7.19E-06

Table IX: Standard deviation of minimum cost for all of the worsening rates evaluated, averaged over all generations (not including the final generation where cost was computed over the full city trace). Note that these data are not normalized as are the data included in the previous figures.

The lines plotted for worsening rate in Figure 23 do not seem to differ significantly, which suggests that worsening rate has little effect on BBO. Also standard deviation of best costs does not change much with worsening probability. Although the standard deviation for worsening probability equal to 1 seems comparatively smaller than the other values, a glance at the scatter plot in Figure 19 suggests that this difference is insignificant.

Finally, we note that the vertical axes on the figures in this section are not comparable, as the minimum cost data for each of the three datasets were normalized independently. Normalization was done because the scale of costs obtained by running BBO for each of these studies is too small to label in the vertical axes of these plots; and the data in the plots were normalized independently because the datasets were collected independently. Nonetheless, comparisons can be made in the case of the scatter plots by keeping note of the scale of the nominal BBO run in each plot, since the Monte Carlo simulations using the nominal BBO parameters were only run once and are used in each of the parameter studies.

We also note that the conclusions drawn in this section are specific to this particular problem. For instance, problems with smooth objective functions that are smooth and characterized by low frequency spectral content may benefit more from an exploitative search strategy, whereas optimization of problems with highly irregular or multimodal objective functions may benefit from explorative search strategies. The optimal search strategy indicated by these results will not apply to all problems in general.

5.3 DACE Parameter Studies

In order to study the study the effect of DACE on a BBO run, we can run Monte Carlo simulations where we vary the parameters of DACE that define how it is used in the framework of BBO. We choose the method proposed in Section 4.5, in which we use DACE to estimate the cost of a fixed number of candidate solutions each generation. We vary the number of sample points, *M*, used to fit DACE models each generation in the range [10, 50, 100, and 200], and the percentage of the population that is estimated using DACE in the range [5%, 10%, 20%, and 50%].We choose simulation parameters for this study based on the BBO parameter study results given in Section 5.2 and the sub-trace results given in Section 2.5 (that is, to use sub-trace 2); these BBO parameters are shown in Table X.

Parameter	Value
Generation count	50
Population size	200
Mutation probability	0.05
Worsening probability	1
Problem dimension	10
Number of elite solutions	2
Simulation drive trace	sub-trace 2

Table X: Nominal BBO parameters for DACE parameter studies.

First, we show surface fits that are quadratic functions of number of DACE sample points and estimated individual percentage. We fit these quadratic surfaces to both the minimum cost and CPU time data of these simulations, and we plot them in Figure 24 and Figure 25. Quadratic surface fits were chosen because they appear to fit the data better than any other kind of elementary surface that was evaluated. We also note that several of these simulations were run using the R2011a version of the MATLAB software with up to 8 cost functions evaluated in parallel, and the rest of the simulations were run using MATLAB R2013a with up to 12 cost function evaluations in parallel. All of the simulations were run on the same computer using the same simulation software. We acknowledge that this is a significant source of systematic error, though only in the CPU time data that we have obtained. We have multiplied the CPU time data obtained in the simulations where up to 8 parallel simulations were run by a factor of 8/12 in order to compensate for the discrepancy in parallel computation speed.

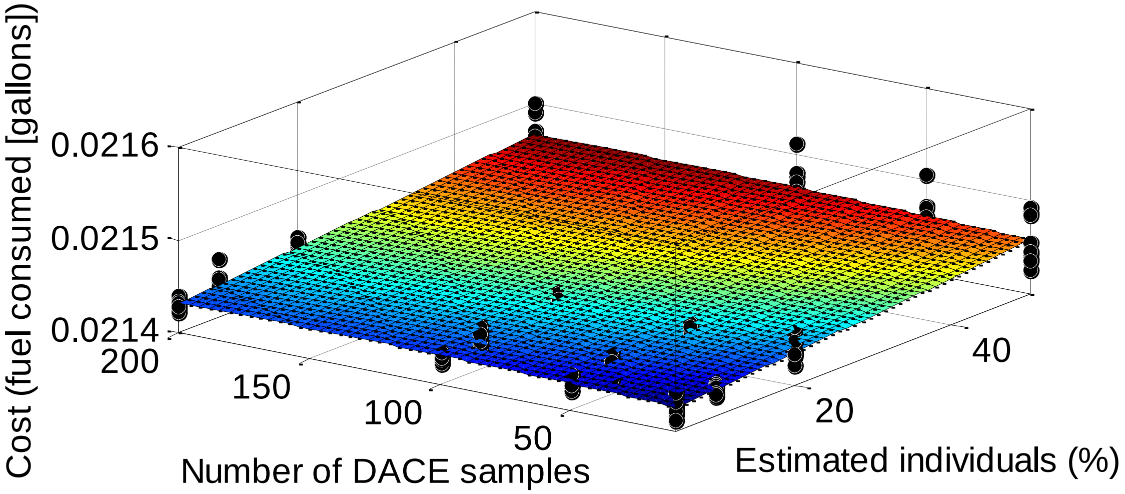

Figure 24: Scatter plot of minimum cost data as number of DACE samples and percentage of estimated individuals are varied, with a quadratic surface fit showing general trends.

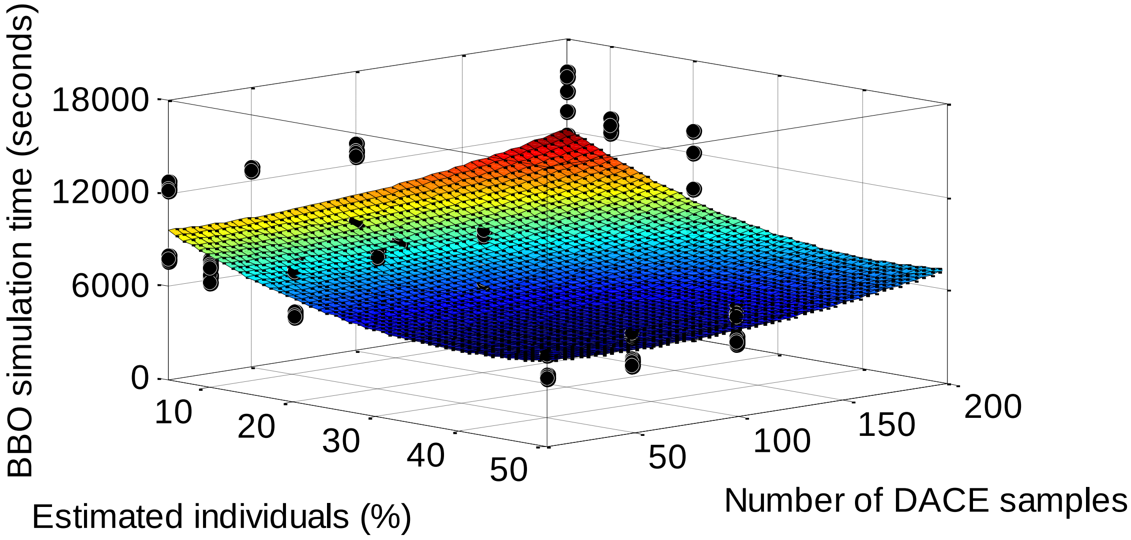

Figure 25: Scatter plot of simulation time data as number of DACE samples and percentage of estimated individuals are varied, with a quadratic surface fit showing general trends.

The relationships we can infer from these figures are that, as number of DACE samples is increased, the minimum cost slightly increases and simulation time also increases. Also, as the percentage of estimated individuals increases, the minimum cost increases, while the simulation time decreases. Because of this, we can use a small number of DACE samples to reduce simulation time and obtain good BBO solutions, and we can also choose a tradeoff between good solutions and simulation time by picking the percentage of the population that we estimate. In our case, however, it seems that the penalty in best solution cost incurred by estimating many individuals is minimal, so DACE can be used aggressively to reduce simulation time while still obtaining good BBO solutions. Finally, we note that the positive correlation between minimum cost and number of DACE samples is a counterintuitive one. More DACE samples implies more estimation effort, which one would expect to improve (i.e., reduce) the best cost obtained with BBO, however, we see the opposite. This result leads us to further examine the error in our DACE models.

We can explain the relationship between number of samples and minimum cost obtained with BBO by viewing the estimation of solutions with DACE as a noisy way of computing cost, and we note that injecting noise into the cost function may actually improve EA performance, especially on harder problems [36], [37]. Cost function noise in an EA can have an effect on the evolution of a population similar to the mutation operator, and thus can be beneficial or detrimental in particular cases [36].

We can also analyze the cost and CPU time data using statistics. We can compute correlation coefficients to determine the relationships between both independent and dependent variables in DACE. Table XI shows the Spearman correlation coefficients between variables, and Table XII shows the null hypothesis probabilities associated with each pair of variables. We have chosen to use the Spearman coefficient because the relationships between the independent and dependent variables are nonlinear (in fact, the relationships are approximately quadratic) as seen in Figure 24 and Figure 25.

Variable	Number of samples	Individuals estimated	Cost	CPU time
Number of samples	1	0	0.2353	0.3035
Individuals estimated	0	1	0.6651	-0.6763
Cost	0.2353	0.6651	1	-0.3938
CPU time	0.3035	-0.6763	-0.3938	1

Table XI: Spearman correlation coefficients between DACE variables.

Variable	Number of samples	Individuals estimated	Cost	CPU time
Number of samples	1	1	0.0045	2.1763E-4
Individuals estimated	1	1	9.7858E-20	1.3834E-20
Cost	0.0045	9.7858E-20	1	1.0470E-6
CPU time	2.1763E-4	1.3834E-20	1.0470E-6	1

Table XII: *p*-values associated with Spearman correlation coefficients between DACE variables.

These tables show a weak positive correlation between number of samples and cost, and a strong positive correlation between the number of estimated individuals and cost. Also, the correlation between number of samples and CPU time is positive, and the correlation between number of estimated individuals and CPU time is negative. This positive correlation between number of samples and cost corroborates our observation from Figure 24; that is, if we decrease the number of samples, cost may go down slightly. Again, this may be explained by considering estimating individuals with DACE as a noisy way of calculating cost, which may improve the minimum cost obtained by an EA [36].

Next, we observe the effect DACE has on the convergence properties of BBO. We show normalized minimum cost plots as a function of the number of generations as we vary the DACE parameters in Figure 26.

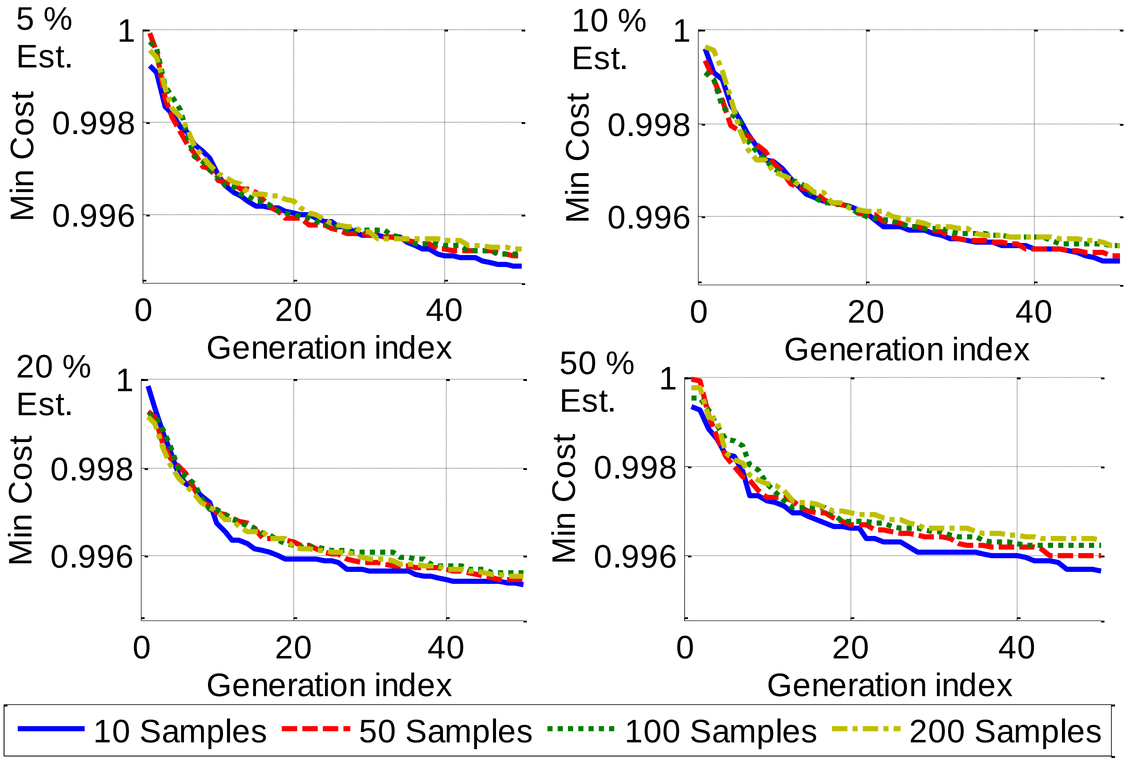

Figure 26: Family of plots in DACE parameters showing normalized minimum cost, averaged over 9 Monte Carlo trials as a function of generation index. Each of the four plots represents a different value of percentage of BBO solutions estimated with DACE, and in each plot, there are four lines, each representing a different value of number of DACE samples.

The BBO populations generally seem to converge within 50 generations, no matter what DACE parameters are used. The data with 50% estimated solutions and 10 samples appears to be an exception to this behavior, but since these data are only an average over 9 Monte Carlo simulations, this is likely due to random chance, and would not be seen if we had averaged over more Monte Carlo simulations. Further, the convergence speed seems generally the same. We also tabulate the standard deviations of minimum costs in Table XIII, and we compute them in the same manner as in the tables in Section 5.2.

Standard Deviation of Minimum Costs

Number of DACE Samples	Percentage of Estimated Solutions			
	5%	10%	20%	50%
10	4.27E-04	2.82E-04	3.71E-04	3.87E-04
50	3.69E-04	4.80E-04	3.97E-04	3.67E-04
100	4.39E-04	3.37E-04	4.04E-04	3.16E-04
200	6.26E-04	3.97E-04	4.82E-04	5.40E-04

Table XIII: Standard deviation of minimum cost for DACE parameters evaluated, averaged over all generations (not including the final generation where cost was computed over the full city trace). Note that these data are not normalized as are the data included in the previous figure.

These data suggest that as we increase the number of samples, the variance of the best costs we obtain with BBO increases; however, there is no apparent relationship between number of estimated solutions and standard deviation of minimum cost. The relationship between number of DACE samples and minimum cost standard deviation may be due to worst case error that results from overfitting. Overfitting means that we may be generating DACE models that are very accurate in the neighborhoods of the samples used to construct the models, but the model error when evaluated far away from these samples may be significantly higher when using a smaller number of samples.

We can examine the difference between using DACE aggressively versus using it conservatively by first noting that we can choose 10 DACE samples without a minimum cost penalty (in fact, the results show that using fewer samples actually improves the best solutions we find with BBO). Next, we can compare the cost and simulation time obtained by running BBO and estimating 5% of the population with DACE, to estimating 50% of the population with DACE. When estimating 5% of the population with 10 DACE samples, the average simulation time is 9641 and the average best cost is 0.02142, whereas when estimating 50% of the population with 10 samples, the average simulation time and best cost are 5476 seconds and 0.02146 gallons respectively. When going from 5% to 50% estimated individuals, the percent increase in cost is 0.1867%, and percent decrease in simulation time is 43.20%. These percent changes should be similar when comparing not using DACE at all versus using it aggressively (e.g., estimating half of the BBO solutions in a run). One's optimization objectives will determine whether this tradeoff is acceptable. One of the future research items we propose is the application of this framework of BBO and DACE to higher fidelity simulation models of the vehicle to further validate the simulation results we get. This simulation may run at 10% of the speed of our current automotive simulation, so instead of saving 3965 seconds by estimating half of the BBO solutions with DACE, we may instead save 39650 seconds. This corresponds to reducing the simulation time of a 27 hour BBO run by more than 11 hours!

Finally, we note that these conclusions are only valid for this problem. For example, one would expect to see different results and to draw a different conclusion when applying DACE and BBO to a problem with an objective function that is more difficult to fit with DACE than our problem. In this case, a greater number of samples may be necessary to achieve enough estimation accuracy for good BBO performance.

5.4 Discussion about BBO and DACE parameters

We have found that the most effective BBO parameters for our problem include a population size that is large enough to contain significant information, but small enough so that computational complexity is not unnecessarily increased; specifically, a good tradeoff appears to be around 200. Also, a mutation rate around 0.05 appears to produce the best solutions to our problem, while worsening rate appears to have an insignificant effect on the solutions we obtain with BBO. We note that population size has a significantly larger correlation with minimum cost obtained in a given BBO run than mutation or worsening rate, so we can consider population size to be the most important BBO variable. We have also found that increasing population size often decreases the minimum cost of solutions obtained during the first generation, but does not change the rate that the BBO population converges. On the other hand, changing mutation rate does not change the minimum cost of solutions obtained for the first generation, though it may have an effect on the convergence rate of the BBO population (an insignificant one in our case), and it has a strong effect on the standard deviation of minimum costs. Finally, worsening rate has an insignificant effect on BBO convergence.

In order to best use DACE with BBO on our problem, we should use a small number of samples (we have obtained good results with 10, though fewer may work as well) and estimate a large number of solutions each generation (we have successfully estimated up to 50% of a BBO population, though estimating more solutions may not incur significantly larger penalties). Doing so reduces computational effort by more than 40% but does not significantly reduce the cost of the best BBO solutions that we obtain. Further, using DACE does not significantly affect the convergence properties of BBO, so we expect to be able to maintain the number of BBO generations, but simultaneously increase the population size and the number of estimated solutions to obtain better solutions (by increasing the amount of evolutionary information contained in the population) while not significantly increasing the computational effort. Future work includes using DACE with BBO in this way.

CHAPTER VI

RESULTS OF CAM TIMING CONTROL OPTIMIZATION WITH BBO

6.1 Introduction

We explore our BBO simulation results in the cam timing optimization problem domain, so that we can determine what particular improvements we have made and what implications these improvements will have in the engine system. Instead of running additional simulations, we choose to use the optimal BBO solutions from the various Monte Carlo simulations of the previous sections, and we note that the differences between the best solutions from run to run are minimal. Table XIV lists the BBO parameters that were used in the optimization run where we obtained our best solution.

Parameter	Value
Generation count	300
Population size	100
Mutation probability	0.02
Worsening probability	1
Problem dimension	10
Number of elite solutions	2
Simulation drive trace	LA4
DACE configuration	not used

Table XIV: BBO parameters used to obtain our best solution.

We note that the parameters in Table XIV are not necessarily the best, since BBO is a stochastic optimization algorithm and the best solution from each BBO run will be a function of random variables; however, our parameter studies given in the previous chapters suggest that we have a better chance of finding good solutions with these parameters than with the others we have tried.

The solution variables (RBF heights) for the best solution that we have found are given in Table XV—the RBF indices correspond to the RBFs as numbered in Figure 3.

RBF Index	Intake adjustment	Exhaust adjustment
1	-14.836	-17.340
2	19.691	17.324
3	-19.729	-19.241
4	19.668	-18.753
5	19.856	-19.619

Table XV: Solution parameters resulting in the lowest cost that we have obtained with BBO.

Notice that many of these RBF heights are close to the limits of the search range [-20, 20]. Generally, if an EA consistently finds solutions close to the search range limits, then the search ranges should be changed, as the optimum that the EA is converging to may be found outside of the range. We have chosen not to adjust our search ranges in this case, because doing so may encourage BBO to generate solutions that have deleteriously large differences between adjacent lookup table values. As stated before, a significant future work direction is to increase the search ranges, establish limits on the changes between adjacent lookup table values, and implement these as constraints in BBO, which may allow us to find better solutions. We also notice that there are combinations of both large negative and positive values in this BBO solution's variables. If a large positive value is applied to the height of an RBF, and a large negative value is applied to an RBF adjacent to the first, this results in the maximum change between adjacent lookup table values that we can apply. Since we are seeing this behavior in our best solution, this indicates that we are already in danger of producing lookup tables that have abrupt changes.

It is also important to examine the differences between our optimal solution and the reference cam timing lookup tables. Figure 27 shows visualizations of the lookup tables before and after BBO.

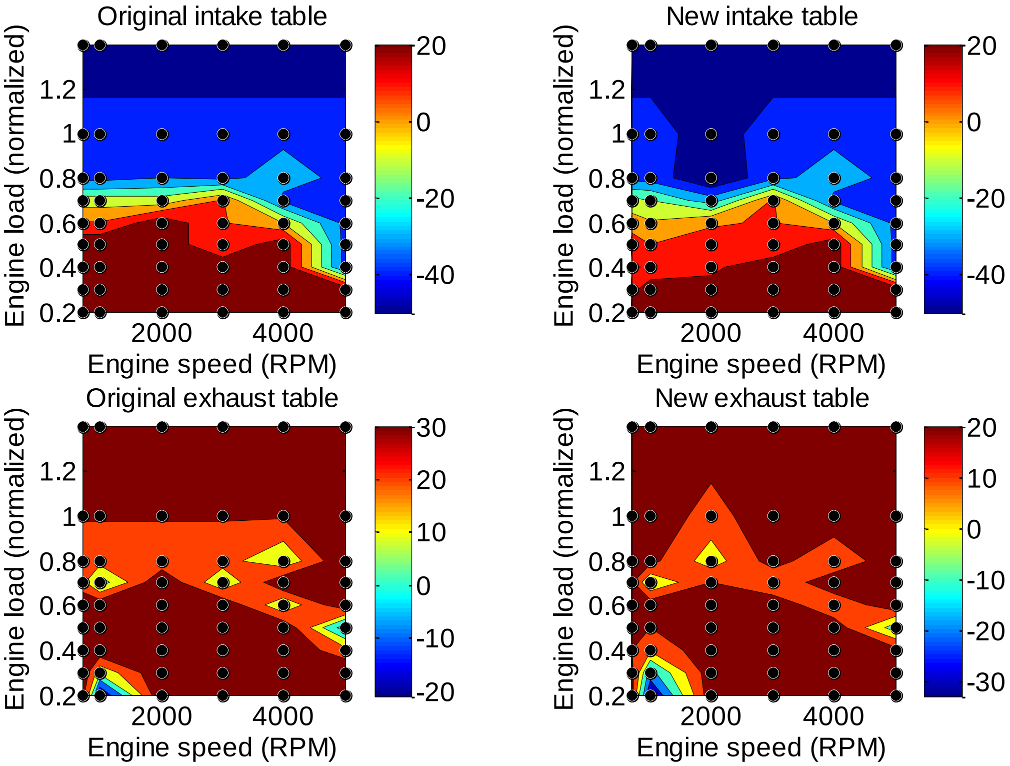

Figure 27: Filled contour plots of the unchanged reference intake and exhaust cam timing lookup tables (on the right, labeled as Original), and the intake and exhaust tables from the best solution obtained with BBO (on the left, labeled as New.) Locations of lookup table values in (engine load, engine speed) are indicated with black circles.

We can observe the difference between adjacent lookup table values by comparing the colors surrounding adjacent black circles in the above plots. The differences between adjacent table elements before and after BBO are fairly similar, so it is unlikely that the differences between adjacent values in the new lookup tables obtained via BBO are excessive and thus likely won't result in ringing or other control problems.

The cost of this best solution is 0.44152 gallons, and the cost obtained by running the simulation with the nominal, unchanged cam timing lookup tables is 0.44919 gallons. This means a decrease of 1.7%. This is a significant improvement that can be made without any changes to the control algorithm or the hardware.

6.2 Examination of BBO Solutions in Problem Domain

Explanations for this improvement in fuel economy with the new table may be attributable to exhaust gas recirculation (EGR) effects. Judging from the distribution of engine load and engine speed data in Figure 3 and Figure 4, we may identify three operating regions of the engine: idle, low speed cruising, and high speed cruising. We define idle as engine load of 0.2, and engine speed of 700 RPM. Also, we define low speed cruising as engine load of 0.4 and speed of 1200 RPM, and we define high speed cruising as a load of 0.6 and a speed of 1500 RPM. In Figure 28, we show the states of the intake and exhaust valves of cylinder 1 as a function of crankshaft angular position for idle, low speed, and high speed cruising, both before and after modifying the tables with BBO.

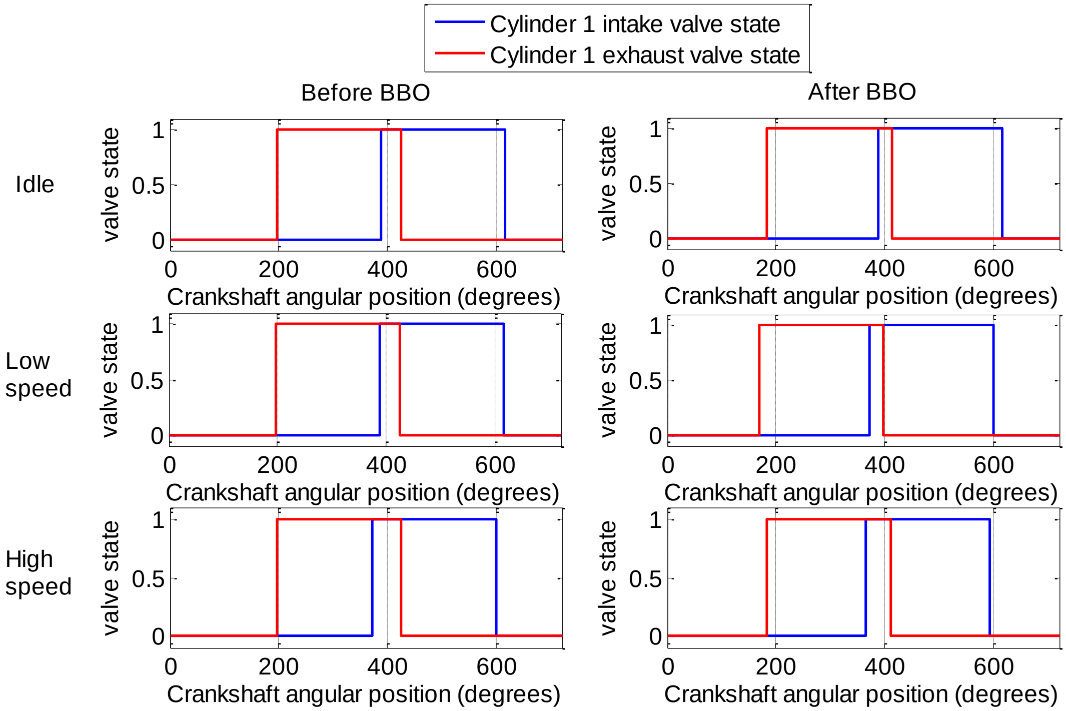

Figure 28: Cylinder 1 valve events for idling, low speed, and high speed operation as a function of crankshaft angle, illustrating valve overlap. The results using the original lookup tables are shown on the left, and the results produced by BBO are on the right. Note that the range of angles shown is [0, 720], because there are two revolutions of the camshaft for each crankshaft revolution.

We can immediately see that the tables produced by BBO have less valve overlap at low vehicle speed, and BBO retards both intake and exhaust valve events significantly, especially at higher vehicle speed. The fuel economy improvement may be attributable to earlier closing of the intake valves, which reduces pumping losses [8]. We can also see exhaust port EGR behavior both before and after optimization with BBO, as there is valve overlap after top-dead-center [29] (which occurs at 360 degrees in Figure 28.) This kind of operation is defined by the intake valve being open for part of the exhaust stroke, during which the high cylinder pressure forces residual gasses into the intake manifold which can also reduce pumping losses. We also note that the reduced valve overlap at idle may be good for reasons besides fuel economy, since the EGR behavior that results from valve overlap reduces the amount of combustible material in the cylinder, and can reduce combustion stability. It is important to reduce EGR at idle, since combustion stability is particularly fragile at idle.

We note that the best solutions obtained by BBO are generally similar, and so we can examine the change that BBO makes to the intake and exhaust tables on average, and what parts of these tables vary the greatest by taking the mean and standard deviation of the tables produced by various BBO runs. In Figure 29, we show the means and standard deviations of the intake and exhaust lookup tables, taken over the best solutions obtained in all of the Monte Carlo simulations given in Section 5.2. We also include contour plots of some exemplary lookup tables from these Monte Carlo simulations in Figure 30.

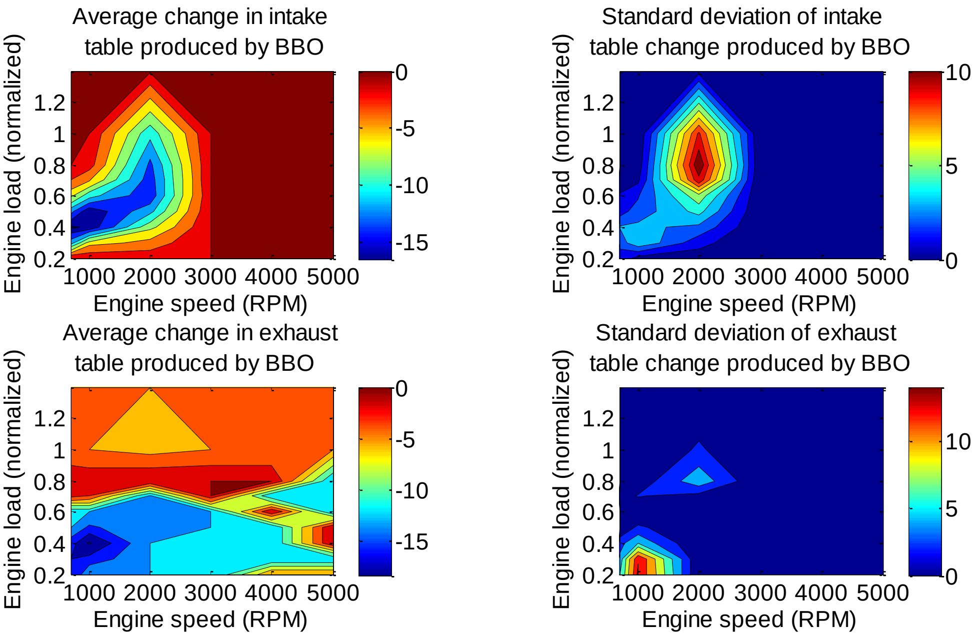

Figure 29: Average changes in intake and exhaust lookup tables, and standard deviation of changes to those tables produced by BBO.

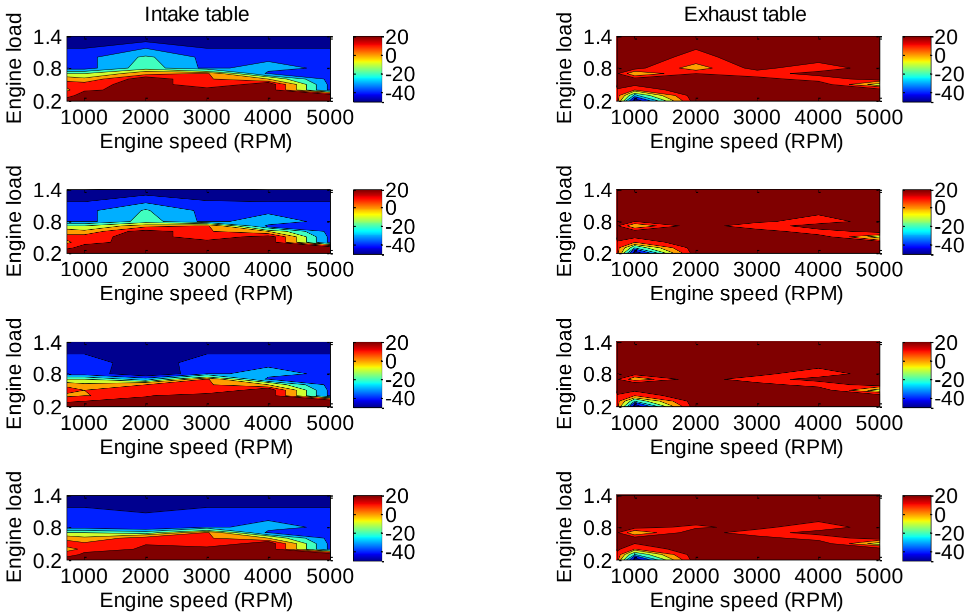

Figure 30: Examples of lookup tables from the best solutions obtained by running BBO several times. Several exemplary intake tables are shown on the left, and several exhaust tables are shown on the right.

First, we note that the differences between tables shown in Figure 30 occur in regions where the standard deviation is high, as shown in Figure 29. The fact that the tables can be different in these ways, and still have similarly low costs, suggests that these changes do not affect cost. This makes sense, because changes in these regions often occur in parts of the lookup table domain that the vehicle does not operate in (as can be seen by the vehicle (load, speed) trajectories in Figure 3 and Figure 4.)

We also note that the changes made to both tables are exclusively negative or zero, this means that BBO is consistently retarding both the intake and exhaust cam timing. Further, for the intake cam, we are most strongly retarding the timing at low to moderate engine load and at low engine speed; this is the state of the engine at idle and when the vehicle starts to move after idle. We can see that the average intake table change is at its highest absolute value and the standard deviation is at its lowest value at idle and low vehicle speed conditions. This indicates that good BBO solutions consistently incorporate this change to the table, and thus, when considering only the range of BBO solutions that we have investigated, this particular change is essential for improving the intake table.

We also see that there is significant retarding of the exhaust cam timing throughout the whole table. We also see that the standard deviation is low throughout most of the table, except at very low engine load and moderate RPM. The engine may not be delivering torque while it is in this state, and so, since modern vehicles incorporate fuel cutoff during engine braking, the state the valves are in during deceleration may matter less. Because we observe a significant absolute value of exhaust timing change and low standard deviation throughout most of the table simultaneously, we can conclude that retarding the exhaust timing is also essential. This is likely the case, because in order to achieve the same degree of exhaust port EGR due to valve overlap while retarding the intake cam to achieve earlier closing of the intake valve, we need to retard the exhaust cam timing as well.

CHAPTER VII

CONCLUSION

7.1 Summary

We have developed a framework for optimizing the controller set-point lookup tables for a dual independent VCT system using BBO and DACE. This framework includes a medium fidelity simulation of the vehicle for evaluating the cost of control solutions, which are parameterized using RBFs. We reduce simulation time by keeping track of the cost of solutions so that we do not need to evaluate their costs more than once in a BBO run, by approximating the full simulation cost by running over a subset of the simulation drive trace, by adjusting our problem parameterization so that changes to the parameters change the controller lookup tables in the region of the vehicle state space that the vehicle most often traverses, by finding BBO parameters that result in the best optimization results, and by estimating solutions with DACE.

The best search configuration for BBO that we have found involves parameterizing the (load, speed) space of lookup table modifications with five RBFs, running BBO with a population size around 200 and mutation probability around 0.05, and using fuel consumption computed over sub-traces with high RMS velocity and acceleration as the cost function for BBO. We have found other BBO variables and other quantities of the simulation drive traces to have negligible effect on the performance of BBO. We have also developed an application of DACE to BBO that takes the numerical conditioning of DACE models into account. We can improve the numerical properties of our DACE models by using sample sets that are well spread out. Also, we can consistently find locally optimal DACE models with fmincon after making the improvements to our DACE fitting algorithm listed in Appendix B.

We have found a BBO solution that results in a 1.7% improvement in fuel economy. The vehicle we are simulating is an SUV with an average annual product of 120,000 units, and an average fuel economy of 22 miles per gallon. If we assume that these improved lookup tables are applied to 120,000 vehicles, each of these cars are driven 10,000 miles annually, the average fuel economy of the vehicle without the improvement is 22 miles per gallon, and that our 1.7% improvement in fuel economy can be extrapolated to apply to all of these vehicles, the reduction in fuel consumed by this group of vehicles will be over900 thousand gallons. Also, since about 9 kilograms of CO_2 are produced for every gallon of gasoline consumed [38], this results in a reduction in CO_2 emissions of 8,100 metric tons. This means a significant reduction in cost from a variety of sources. Not only will consumers save on fuel, but the cost associated with environmental impact will be reduced as well.

7.2 Future Work

Future work may include running BBO on higher fidelity vehicle simulations to find optimal cam timing lookup tables. It is likely that the locations of good control solutions within the search space will change when using a more accurate simulation, and a more accurate simulation will help validate our simulation results. Other future work may include systematic approaches to search space parameterization or BBO parameter optimization. Finally, the BBO implementation we have produced for cam timing optimization may be applied to a variety of other computationally expensive optimization problems. Another direction of future work is to use multiobjective BBO to simultaneously optimize engine control for a fuel economy, emissions, and power, and the control of other automotive subsystems can be optimized with BBO.

REFERENCES

[1] C. S. Draper and Y. T. Li, *Principles of Optimalizing Control Systems and an Application to the Internal Combustion Engine*. New York, NY: American Society of Mechanical Engineers, 1951.

[2] Society of Automotive Engineers, *Automotive Fuel Economy*. Warrendale, PA: Society of Automotive Engineers, 1976.

[3] Congress of the United States: Office of Technology Assessment, *Improving Automobile Fuel Economy*. Washington, DC: US Government Printing Office, 1991.

[4] W. B Ribbens, "Electronic Engine Control," in *Fuel Economy: In Road Vehicles Powered by Spark Ignition Engines*. New York, NY: Plenum Press, 1984, pp. 419-447.

[5] J. Zhao and X. Min, "Fuel Economy Optimization of an Atkinson Cycle Engine Using Genetic Algorithm," *Applied Energy*, vol. 105, no. C, pp. 335-348, 2013.

[6] M. Kim, T. Hiroyasu, and M. Miki, "Multi-Objective Optimization of Diesel Engine Emissions and Fuel Economy Using SPEA2+," in *GECCO '05 Proceedings of the 2005 conference on Genetic and evolutionary computation*, Washington, DC, 2005.

[7] B. Wu, R. G. Prucka, Z. S. Filipi, D. M. Kramer, and G. L. Ohl, "Cam-phasing optimization using artificial neural networks as surrogate models–fuel consumption and NOx emissions," in *SAE World Congress*, Detroit, MI, 2006.

[8] M. Sellnau and E. Rask, "Two-Step Variable Valve Actuation for Fuel Economy, Emissions, and Performance," in *SAE World Congress*, Detroit, MI, 2003.

[9] J. Kennedy, C. Eberhart, and Y. Shi, *Swarm Intelligence*.: Morgan Kaufmann, 2001.

[10] D. Simon, *Evolutionary Optimization Algorithms: Biologically-Inspired and Population-Based Approaches to Computer Intelligence*. Hoboken, NJ: John Wiley & Sons, 2013.

[11] F. Pinel, G. Danoy, and P. Bouvry, "Evolutionary algorithm parameter tuning with sensitivity analysis," in *Proceedings of the 2011 international conference on Security and Intelligent Information Systems*, Warsaw, Poland, 2011.

[12] K. De Jong, "Parameter setting in EAs: a 30 year perspective," in *Parameter Setting in Evolutionary Algorithms*.: Springer Berlin Heidelberg, 2007, pp. 1-18.

[13] J. Halloran and A. Erdemir, "Adaptive Surrogate Modeling for Expedited Estimation of Nonlinear Tissue Properties Through Inverse Finite Element Analysis," *Annals of Biomedical Engineering*, vol. 39, no. 3, pp. 2388-2397, September 2011.

[14] J. Sacks, W. Welch, T. Mitchell, and H. Wynn, "Design and Analysis of Computer Experiments," *Statistical Science*, vol. 4, no. 4, pp. 409-935, 1989.

[15] G. Wang and S. Shan, "Review of Metamodeling Techniques in Support of Engineering Design Optimization," *ASME Transactions, Journal of Mechanical design*, vol. 129, no. 4, pp. 370-380, 2007.

[16] K. Elsayed and C. Lacor, "Robust Parameter Design Optimization Using Kriging, RBF and RBFNN with Gradient-Based and Evolutionary Optimization Techniques," *Applied Mathematics and Computation*, vol. 236, no. 1, pp. 325-344, June 2014.

[17] G. Matheron, "Principles of Geostatistics," *Economic Geology*, vol. 58, no. 8, pp. 1246-1266, 1963.

[18] D. Simon, "Biogeography-based optimization," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 702-713, 2008.

[19] D. Simon, M. Ergezer, D. Du, and R. Rarick, "Markov Models for Biogeography-Based Optimization," *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, vol. 41, no. 1, pp. 299-306, February 2011.

[20] H. Ma and D. Simon, "Analysis of Migration Models of Biogeography-Based Optimization Using Markov Theory," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 6, pp. 1052-1060, September 2011.

[21] D. Simon, R. Rarick, M. Ergezer, and D. Du, "Analytical and Numerical Comparisons of Biogeography-Based Optimization and Genetic Algorithms," *Information Sciences*, vol. 181, no. 7, pp. 1224-1248, April 2011.

[22] D. Simon, "A Probabilistic Analysis of a Simplified Biogeography-Based Optimization Algorithm," *Evolutionary Computation*, vol. 19, no. 2, pp. 167-188, June 2011.

[23] G. Thomas, P. Lozovyy, and D. Simon, "Fuzzy Robot Controller Tuning with Biogeography-Based Optimization," in *24th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems*, Syracuse, NY, 2011.

[24] T. Wilmot et al., "Biogeography-Based Optimization for Hydraulic Prosthetic Knee Control," in *Medical Cyber-Physical Systems Workshop*, Philadelphia, PA, 2013.

[25] A. Bhattacharya, "Biogeography-Based Optimization for Different Economic Load Dispatch Problems," *IEEE Transactions On Power Systems*, vol. 25, no. 2, pp. 1064-1077, 2010.

[26] H. Langouët, L. Métivier, D. Sinoquet, and Q. Tran, "Optimization for Engine Calibration," in *EngOpt 2008 - International Conference on Engineering Optimization,* Rio de Janerio, 2008, pp. 1-5.

[27] K. Atashkaria and N. Nariman-Zadeh, "Modelling and Multi-Objective Optimization of a Variable Valve-Timing Spark-Ignition Engine Using Polynomial Neural Networks and Evolutionary Algorithms," *Energy Conversion and Management,* vol. 48, no. 3, pp. 1029–1041, March 2007.

[28] C. Lin, H. Peng, and J. Grizzle, "A Stochastic Control Strategy for Hybrid Electric Vehicles,"

in *Proceeding of the 2004 American Control Conference*, Boston, 2004, pp. 4710-4715.

[29] J. Meyer. (2007, March) The Ohio State University. Department of Mechanical Engineering Honors Theses, Engine Modeling of an Internal Combustion Engine with Twin Independent Cam Phasing. [Online]. http://hdl.handle.net/1811/24538

[30] B. Baxter. (1992, August) Cambridge University. Department of Mathematics Dissertation, The Interpolation Theory of Radial Basis Functions. [Online]. http://www.cato.tzo.com/brad/papers/thesis.pdf

[31] US EPA. (2012, August) Dynamometer Drive Schedules. [Online]. http://www.epa.gov/nvfel/testing/dynamometer.htm

[32] Y. Jin and J. Branke, "Evolutionary Optimization in Uncertain Environments--A Survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303-317, 2005.

[33] D. Jones, M. Schonlau, and W. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455-492, 1998.

[34] M. D. McKay, R. J. Beckman, and W. J. Conover, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239-245, 1979.

[35] W. Dembski and R. Marks, "The Search for a Search: Measuring the Information Cost of Higher Level Search," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 14, no. 5, pp. 475-486, April 2010.

[36] J. Branke and C. Schmidt, "Selection in the Presence of Noise," in *Genetic and Evolutionary Computation - GECCO 2003*, Chicago, IL, USA, 2003, pp. pp 766-777.

[37] C. Qian, Y. Yu, and Z Zhou. (2013, November) Analyzing Evolutionary Optimization in Noisy Environments. arXiv preprint: 1311.4987.

[38] U.S. Energy Information Administration. (2014, April) How much carbon dioxide is produced by burning gasoline and diesel fuel? [Online]. http://www.eia.gov/tools/faqs/faq.cfm?id=307&t=11

[39] MathWorks. (2014, January) Moore-Penrose pseudoinverse of matrix - MATLAB pinv. [Online]. http://www.mathworks.com/help/matlab/ref/pinv.html

[40] G. W., Aschenbach, C. R. Balich. (2004) The Gasoline 4-Stroke Engine for Automobiles. [Online]. http://www3.nd.edu/~msen/Teaching/DirStudies/Gas4Stroke.pdf

[41] Ford Motor Company. (2010, February) Twin Independent Variable Camshaft Timing (Ti-VCT) [Technology Flyer]. [Online]. http://www.at.ford.com/news/Publications/Publications/Ti_VCT_Engine_Technology_FS.pdf

[42] A. Ghazimirsaied, S. Jazayeri, and A. Shamekhi, "Improving Volumetric Efficiency Using Intake Valve Lift and Timing Optimization in SI Engine," *International Review of Mechanical Engineering*, vol. 4, no. 3, pp. 244-252, March 2010.

APPENDIX A: LOOKUP TABLE INTERPOLATION METHODS

This appendix discusses evaluation of two different schemes for producing actuator control signals given the original lookup tables and the sum of the RBFs from a BBO cam timing solution. We begin by identifying two different ways of applying the RBF surfaces from BBO—in the first, we sample the RBF surface at each point in the lookup table domain, add these samples directly to the corresponding indices of the lookup table, and then finally round, and saturate (i.e., truncate) the lookup table values so that they lie within the range of values allowed for the actuators. This method produces a new lookup table that can be applied directly in the ECU, and the ECU linearly interpolates between the values of the new, optimized lookup table. We refer to this scheme as “interpolating later.” In the second method, we evaluate the RBF in the ECU at run time, and add these RBF values to values interpolated from the original, unchanged lookup table. We call this scheme, “interpolating first.”

We identified the question of whether one approach is better for control than the other as an important one to answer in our research, because details of the problem formulation such as this can have a significant effect on the kind of solutions that can be found using an EA such as BBO. The results of this investigation are given in this appendix. We note that, to simplify this examination, we do not apply rounding and saturation to the interpolating later scheme as we have used it with BBO.

We begin by considering the cases under which both approaches are equivalent. Both cases interpolate between adjacent lookup table values using bilinear interpolation. Let the domain of the lookup table be in the space, [l, r.] The formula that gives the interpolated lookup table values within the range $l \in [l_1, l_2]$, and $r \in [r_1, r_2]$, is then given by

$$f(l,r) = L^{\mathrm{T}}(T)R \qquad \text{(A.1)}$$

where

$$L = \begin{bmatrix} l - l_1 \\ l - l_2 \end{bmatrix};\ R = \begin{bmatrix} r - r_1 \\ r - r_2 \end{bmatrix};\ T = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} \tag{A.2}$$

and t_{11}, t_{12}, t_{21}, and t_{22} are the lookup table values at $[l, r] = [l_1, r_1]$, $[l_1, r_1]$, $[l_1, r_1]$, and $[l_1, r_2]$ respectively.

When interpolating later, the formula for the output (i.e., interpolating over the new lookup table) is given by

$$f_1(l, r) = L^{\mathrm{T}}(G + T)R;\ G = \begin{bmatrix} g(l_1, r_1) & g(l_1, r_2) \\ g(l_2, r_1) & g(l_2, r_2) \end{bmatrix} \tag{A.3}$$

where $g(l, r)$ is the RBF surface found using BBO.

When instead interpolating first, the formula for the control output (the sum of the RBF surface calculated at runtime and the linearly interpolated unaltered table) is then given by

$$f_2(l, r) = L^{\mathrm{T}}(T)R + g(l, r) \tag{A.4}$$

Then, to make $f_1 = f_2$, we must have

$$L^{\mathrm{T}}(G + T)R = L^{\mathrm{T}}(T)R + g(l, r) \tag{A.5}$$

$$L^{T}(G + T)R - L^{T}(T)R = g(l, r) \tag{A.6}$$

$$L^{T}\big((T + G)R - TR\big)R = g(l, r) \tag{A.7}$$

$$L^{T}(G)R = g(l, r) \tag{A.8}$$

Equation A.8 means that the only way the two approaches are equal is if the RBF surface is identical to bilinear interpolation defined by the elements of G. We suspect that this is not possible in general. However, if we do not restrict $g(l, r)$ to be a function in Gaussian bases, but instead make it a linear function of some other bases or even a nonlinear function, or if we use a different kind of interpolation, then there may be other cases where $f_1 = f_2$.

We note that we chose to use approach of interpolating later (i.e., adding RBF surface values to the lookup table, then interpolating,) because the BBO results can then be used to generate optimal lookup table values which can immediately be put into the automobile. However, it would be good to consider this issue further in the future.

One future work approach may be to compare the spectral content of control signals generated with either approach. We hypothesize that our approach of interpolating later sacrifices the ability to reproduce some spectral content (and thus places greater limits on the kinds of control solutions we can produce) because we are throwing away a certain amount of information when we take the continuous RBF surface and use only samples of it for adding to the lookup table. This step is essentially discretization, and the Nyquist theorem states that we can recover frequencies from the discretized version of the RBF surface up to one half of the sampling frequency, though the fact that the tables are in two dimensions, and the spacing between table elements is nonuniform makes thus difficult to analyze in the frequency domain.

Because of this, we have not mathematically determined what kind of frequency content we are sacrificing, though we suspect that the ability to produce additional high frequency content in the controls is not necessary, because cam phasors like the one we are simulating have relatively slow dynamics (e.g., it may take several seconds for the VCT control system to track from one angle set point to another.)

APPENDIX B: IMPROVING DACE NUMERICAL CONDITIONING

In this appendix, we discuss strategies we have taken to improve the numerical properties of our DACE modeling approach. Our first strategy for improving the numerical properties of DACE is to use the MATLAB implementation of the Moore-Penrose pseudoinverse instead of the normal matrix inverse, and the pseudodeterminant instead of the determinant. The pseudoinverse and pseudodeterminant are useful for cases where the correlation matrix, *R*, is ill-conditioned [10].

The pseudoinverse is computed, in our case, by finding the singular value decomposition and setting all singular values less than a given tolerance to zero [39]. This forces the system to be underdetermined, and we can subsequently find a minimum norm solution for the DACE cost function estimate. Also, we define the pseudodeterminant to be the product of all eigenvalues greater than a given tolerance.

Our second strategy is to maximize the log-likelihood instead of the likelihood during the DACE fitting process. In our experience, computing the likelihood in cases with a large number of samples sometimes results in floating-point overflows and underflows, whereas the log-likelihood does not suffer from this problem. This is because the dynamic range of likelihood values that we can represent with the log-likelihood function is much greater than the dynamic range we can represent with the likelihood function itself. Dynamic range is the difference (in decibels) of the maximum and minimum values that can be represented in a particular kind of variable, and the MATLAB expression

```
10*log10(realmax) - 10*log10(realmin)                    (B.1)
```

can be used to compute the dynamic range that can be represented by the double precision variables that we use in our computations—this expression evaluates to approximately 6159 dB. This is the dynamic range of likelihood values we can normally represent. Note that the quantities realmax and realmin are the largest and smallest non-zero positive numbers that can be represented in the IEEE double precision floating point standard respectively. A MATLAB expression to evaluate the dynamic range of likelihood values that we can represent with log-likelihood values contained within double variables can be given by

```
10*log10(exp(realmax)) - 10*log10(exp(realmin))
```

(B.2)

which simplifies to

```
10/log(10) * (realmax - realmin)
```

(B.3)

However, this number itself cannot be represented by a double, since the expression evaluates to $7.807 \cdot 10^{308}$ dB, which is over 4 times greater than realmax. Our manual simulation tests have shown that this approach provides more than enough dynamic range for likelihood maximization and thus solves the overflow/underflow problem.

Our third strategy is to improve our choice of initial search conditions in θ and p parameters for fmincon. We have found that our first approach of setting the initial elements of θ and p to be the same is a poor one, because the search space in θ and p around that initial guess often appears to be too flat for our derivative-based fmincon optimization algorithm. We originally set θ = [0.1, 0.1, ..., 0.1] and p = [1.5, 1.5, ..., 1.5]. Our current choice of initial guess, θ = [0.1, 1.0, ..., $10^{(N-2)}$] and p = [1.5, 1.5, ..., 1.5] appears flat to fmincon much less often.