ABSTRACT

PROPST, MICHAEL DAVID. Applying Linear Regression and Neural Network Meta-Models for Evolutionary Algorithm Based Simulation Optimization. (Under the direction of Dr. Jeffrey Joines.)

The advances in computing power over the last decade have led to an increase in the use of simulation programs and their complexity to model real world optimization. An experimental design is often used to determine the effects varying parameters have on the desired output. The user is usually trying to optimize or minimize a series of outputs. However, these problems often have a large number of variables or parameters that can be changed with a wide range of values increasing the complexity of the model. As these simulation models become more complex they become computationally expensive to run. Most of these problems are non-linear and based on the inherent variability in real-world applications and may not have a true optimal solution. Evolutionary algorithms are a class of computational optimization techniques that are based on the principles of evolution and harness the power of the computer to solve a problem. The application of evolutionary search techniques as a simulation optimization technique has yielded promising results. However, the algorithm can take a long time evaluating just one set of decision variables owing to replications and computational time of one simulation run and not to mention the sheer number of different sets that have to be evaluated to find good solutions for these complex problems. Linear regression and neural-network meta-models can be used to generate a meta-model of the simulation. Evaluating the meta-model is very fast as compared to the

simulation model. Therefore, this thesis combines the use of evolution algorithms, simulation models and meta-models to produce a more efficient simulation optimization technique. The two types of meta-models are tested to determine their effectiveness as a meta-modeling technique and the overall effectiveness of finding the best solution.

Applying Linear Regression and Neural Network Meta-Models for Evolutionary Algorithm Based Simulation Optimization

by
Michael David Propst

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Textile Engineering

Raleigh, North Carolina

2009

APPROVED BY:

Dr. Jeffrey A. Joines Committee Chair	Dr. Timothy Clapp
Dr. Jon Rust	Dr. Jeff Thompson

DEDICATION

I would like to dedicate this paper to Jon Rust. A man whose influence on my life I could not begin to explain. He found a confused boy and worked tireless to unlock his potential and had faith when others didn't. Without whose small but effective words of motivation I would have never considered graduate school yet alone coming back to complete this project four years later. His guidance I am grateful for, inspiration I am thankful for, and his friendship I will always cherish.

I would like to thank Jeff for having patience and not giving up when others would have and digging into the archives to finish this.

And to mom….well for being mom and always understanding and but somehow pushing.

"Gain the admiration of intelligent men and the respect of great men"

BIOGRAPHY

Michael David Propst – was born in North Carolina and raised in Concord. He found his way to NC State via Northwest Cabarrus High School. He received his B.S. in Textile Engineering from North Carolina State University in Dec of 2002. He then proceeded directly to work on his M.S. in Textile Engineering from the same university. Upon completion of his coursework he began working for Abercrombie & Fitch Trading Company as a Quality Engineer. During which time he spent numerous months in South Asia and South East Asia in garment production optimization eventually completing his M.S. thesis in August of 2009.

ACKNOWLEDGMENTS

I want to acknowledge Dr. Jeff Joines who helped guide me through this journey and was with me every step of the way; especially the late steps.

I would like to thank Angie Brantley who made sure all of my paperwork was submitted properly and on-time. She was instrumental as I was not in NC during the completion of this.

I would like to thank my committee members who were exceptionally patient with the constant rescheduling and for Dr. Thompson who came on at the end when we needed his help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

Maximizing resources and/or minimizing costs are essential for any company to excel in the increasingly competitive international economy. In designing, creating and/or improving a product or process of any sort it is important to be able to optimize the system or product with the appropriate set of inputs to increase the profitability of the product. Also, "any abstract task to be accomplished can be thought of as solving a problem, in turn a search through a space of potential solutions." [Michalewicz, 1996] Examples can range from determining the optimal set of parameters for a series of machines to produce the best quality for a given price, designing the optimal layout of a manufacturing floor or airport to optimizing a 401K portfolio so one can retire earlier.

The output of these processes commonly relies on many variable inputs that are often interdependent with one another and can range from discrete to continuous inputs, to qualitative inputs, to statistical distributions. Many of these systems to be optimized do not have a closed form (i.e., a set of equations to optimize) owing to the complexity of the system. Computer simulation is commonly used to predict the output of these processes. This involves developing a model of the system in a simulation modeling language that takes the inputs that drive the particular process and produces a series of output metrics of the system. These inputs can be constants (e.g., the number of workers, the number of check out

people, baggers, etc.) or stochastic. Stochastic inputs (e.g., processing times of jobs at a machine, the time between arrivals, the number of items selected at a store, the number of people who arrive at a store) are random and occur based on a distribution. These inputs increase the complexity of the simulation. For example, consider a grocery store simulation where one is trying to determine the optimal number of baggers and checkout clerks needed during each time period the grocery store is open. A model of the system can be developed by determining the statistical distributions of the arrival rate of customers to the store, the length of stay, the check out processing times, the number of items purchased, etc. Executing the simulation model can take a substantial amount of time when replications are required; which is common when dealing with stochastic simulation models to produce a more accurate picture of the true outcome. Therefore, it is often the case that only a few different sets of inputs are tried and the best system is taken from this subset. In order to maximize the efficiency of a process, simulation is commonly used in conjunction with single variable experimentation, which entails changing one variable at a time and observing the output. This method of experimentation is devoid of intelligent planning and any explanation of interaction effects of any order. This method also limits the search space to that conceived by the searcher and thus based on experience, intuition, or more commonly no data at all.

Very little research has been performed in optimizing simulations as opposed to general mathematical models since no closed form exists. It is impossible to generate search gradients form a black-box function. Traditional optimization methods like gradient descent

do not work in this arena due to the stochastic output. Genetic Algorithms(GA) are a part of a field of computational optimization methods that do not rely on gradients to perform the search and are independent from the evaluation function in making search decisions. These algorithms are known for their ability to search a large search space very efficiently consisting of hundreds or even thousands of variables to achieve an optimum solution. This technique utilizes evolutionary techniques that mimic natural evolution, which include random mutation, simulating mating or crossover, and survival of the fittest. This enables the program to converge to an "optimal solution" while thoroughly exploring the solution space. The term "optimal solution" is meant to represent a very good point, which may or may not be the true optimal of the system due to the nature of GAs.

1.1 Opportunity

Researchers have used various techniques to determine the best set of inputs that will optimize a computer simulation, which in term should optimize the true system [Androitter 1998]. These have been found to be effective in computing a good solution for the optimization of the process. The genetic algorithm can be thought of as a directed random search and during the search process it often creates children (i.e., new solution points) that are poor or away from the best solution (i.e., creating random points to drive the search). The expensive nature of simulation makes the technique of trying random points in a

simulation computationally expensive to determine the optimum of the system. Also, in order to limit the stochastic nature of the output, the simulation is run multiple times for a particular set of variable inputs into the simulation which produces a stochastic output to find a value closer to the mean and thus increasing the evaluations of a single point. The GA maintains a population of solutions which can be thought of as short term memory since this represents a small snapshot of the space which can be lost from generation to generation.

There is an opportunity to increase the computational efficiency of the program by creating a mathematical model of the search space from either the historical data (i.e., starting population) or a planned experiment. Creating a mathematical model of the process from the data to some degree of error will allow a simple mathematical equation for the GA to optimize which will dramatically decrease the amount of computational effort needed to evaluate an individual in a population.

1.2 Approach

The goal is to determine the most efficient way to optimize a simulation model utilizing a meta-model. The approach will determine the best way to determine the meta-model and how to combine it with the actual simulation model and genetic algorithm.

There are many methods available that generate a mathematical model from a data set containing explanatory variables and their outputs. This data can commonly come from historical data representing known solutions, or it can be generated from a simulation itself.

Some of these techniques output a linear model, such as ordinary least squares regression, and some produce a non-linear model, such as neural network modeling techniques. The goal of this research is to fuse one or more of these modeling techniques with the simulation and genetic algorithm searching techniques. These modeling techniques will be used to create a model of the simulation solution data space, or meta-model, which the genetic algorithm searching technique will use to evaluate some of its new solution points. This will give a rough estimate as to the quality of this solution and it is a cheap and inexpensive way to determine if a new solution point should be kept in the population. Even a slight decrease in the amount of simulations that have to be run to complete the optimization can dramatically decrease the amount of time necessary to run the optimization. When should the meta-model be used for evaluation? How often should it be used for evaluation? Under what circumstances should the meta-model be used? Which modeling technique is best? Which modeling technique produces the least error? These are all questions that the research will attempt to answer.

Numerous approaches exist that can be used to create a model for the genetic algorithm to work on. There also many questions associated with these techniques. These questions will be discussed with each technique in Chapter 2. The approaches explored in this paper pertain to the fields of statistical theory and neural computing. Many options and techniques in statistical theory are explored along with a back propagation technique derived from the neural computing field. Chapters 3 and 4 discuss a comparison between generating meta-

models that are created through an adopted stepwise regression program as well as a back propagation neural networking algorithm. These algorithms are executed on both deterministic and stochastic optimization problems of varying complexity. Chapter 5 summarizes the findings including ideas for further research and suggestions upon the direction.

2 Literature Review

The computationally expensive nature of simulations involving a large number of input variables has fueled the proposition of many different methods for determining the optimal values for those input variables to find the best solution. Different modeling techniques exist as a means to replace the computationally expensive simulation with an analytical based model representation. This would allow the model to be accessed as opposed to the simulation itself, thus higher sampling optimization methods to be utilized can properly explore the possible solution space.

2.1 Simulation

Simulation is a powerful tool used for modeling complex systems. It is used to model new system designs, proposed system changes, and retro-fitting existing systems. Simulation is a tool for aiding decision-making. A simulation model represents time and changes that occur in the system or process over time. The majority of simulations are discrete, meaning that changes occur only at discrete points in time; which is in contrary to continuous simulations where changes occur continuously. Discrete event simulation models are less complex than continuous simulation and easier to model. Discrete event simulation models are based on concepts of events, activities, processes, and states; which are commonly explained as events that trigger other events or a process of events. The state of process in the model is

commonly represented by a vector containing variables that describe the state of the system at any time. This vector is referred to as the model state [Carson, 2004].

Similar to other sciences, computer simulation contains its own vocabulary. An event is an occurrence that changes the models state instantly [Carson, 2004]. An example of an event is the arrival of a customer at a checkout lane in a grocery store. An activity is a duration of time, such as the amount of time necessary for the checkout clerk to total the cost of the customers good. Events and activities are commonly modeled using a probability distribution derived theoretically or from historical data [Carson, 2004]. Distributions that can commonly be applied to discrete simulation are the Gaussian (Normal), Poisson, and Geometric distributions. Entities in simulations are objects in the model. Dynamic entities refer to entities created at time points, such as the completion of a customer check-out. Resources provide services to dynamic entities, such as checkers, baggers, etc. Simulations commonly contain stochastic inputs, such as the arrival of customers to a grocery store and the amount of time that they spend shopping before proceeding to the check-out. These inputs are modeled using a probability distribution. These variable stochastic inputs generate a variable stochastic output, creating a distribution or any single configuration of the system or process [Carson, 2004].

Simulation models are used to represent a number of different system or process configurations. This is done by varying the input parameters associated with the simulation model, such as the number of workers, machines, or speed of a transaction [Carson, 2004].

These models can be used to determine the best configuration of the output parameters over the finite sets of configurations. However, the size of the input parameter space, the number of variables, the range of those variables, and the types of variables can make a trial and error method ineffective.

Since simulations are used to model processes that cannot be modeled analytically it is necessary to seek out an alternative way to optimize the simulation. Simulation optimization refers to the search for the combination of configurable variables to maximize or minimize the desired output. This output is commonly the performance of the system or process. Such complex problems are commonly found in manufacturing and supply chain management. The stochastic nature of such problems prevents simple analytical modeling and optimization techniques [Olafsson 2002].

2.2 Simulation Optimization

Simulations are a tool to aid in decision making; not a decision making tool. They are considered a tool to answer "What If" questions [Magoulas 2002]. Simulations can be used to determine the state of controlled inputs and their effects on the output [Bowden 1998]. Simulation optimization can be viewed as finding a combination of input parameters that yields the greatest output [Humphrey]. In order to achieve this optimum output a simulation needs an external operator of some sort to search for the optimum values of the input

parameters. Stuckman et al (1991) divide simulation optimization into three categories. The first being one that utilizes “trial and error” methods. This is randomly varying inputs in an effort to achieve a better performing outcome. The second is systematically varying parameter input and observing the effect on the output (e.g., design of experiments, response surface methodology (RSM)). The goal is to ascertain some correlation between the input changes and the final outcome. The third category is an automated simulation optimization approach [Magoulas 2002].

Discrete decision variables refer to the system or process whose sample space is finite. The optimization methods for discrete decision variables are dependent of the size of the variable space and the complexity of the problem. This is normally determined by whether or not it is feasible to simulate the entire sample space. It is quite common that it is impossible to simulate the complete sample space. For the problems where it is impossible to simulate the entire sample space a technique must be employed to determine which points to sample and thus simulate. Subset selection or screening attempts to reduce the feasible region so as to make it more manageable during the optimization process. If the output space is large and complex then the previous two methods fail to adequately explore the sample space and thus fail to provide a solution with reasonable confidence [Olafsson].

Considerable research has been done in the third category. The ever increasing complexity of simulation problems in conjunction with increasing computing power allow for utilization of automated optimization approaches. These methods need to have a

stochastic component to prevent the search algorithm from getting stuck in local minimum/ maximums. With stochastic based search operators there is no guarantee of finding the true optimal value; only a potentially good one. There are a number of computational optimization techniques (random restart, response surface methodology, Simulated Annealing, Genetic Algorithms, Particle Swarm Optimization, Stochastic Approximation).

2.2.1 Simulated Annealing

This method is inspired by the thermodynamics process of annealing of metals in physics where metal is cooled, reheated, cooled again, etc. to reach the minimum equilibrium state. Simulated annealing explores solutions in a neighborhood and evaluates their fitness. The program is allowed to search for solutions in neighborhood of a lesser value. This prevents the method from converging on a local minimum. Whether or not the lesser valued neighborhood is explored is based on a function utilizing a random number and the amount of time it has been searching. The longer the algorithm has been searched the less likely it is that it will be allowed to explore the lesser valued neighborhood [Avello 2004].

2.2.2 Particle Swarm Optimization Method

Particle swarm optimization is another computational probability meta-heuristic that mimics the behavior of a “bird’s flock” in that social information sharing takes place. Individuals can profit from their own experience as well as those of the other solutions in the “swarm”. Each potential solution in the “swarm” basis the next move on its experience (i.e., direction)

as well as the experience (i.e., direction) from other potential solutions in the "swarm" [Magoulas 2002].

2.2.3 Genetic Algorithms

Genetic algorithms (GAs) mimic Darwin's evolution process to move throughout the space. Since this study utilized this meta-heruistic as the global optimization strategy it will be discussed in detail in Section 2.3.

2.2.4 Stochastic Approximation

Stochastic approximation is the process of moving in the direction of greatest ascent. There is no analytical function from which to determine this direction, thus the gradient is estimated by choosing points at a finite distance form the current solution in the solution space to approximate the gradient. These points can either be chosen in a single variable direction to the current solution and the gradient projected in the other direction or points can be chosen from both sides. It has been proven that stochastic approximation will converge asymptotically given certain conditions and a sufficiently slow decrease in the step size. A draw back to this method is that with each step of the stochastic approximation, $n+1$ simulations are needed for the one sided technique and $2n$ simulations are need for the two sided technique; n being the number of configurable variables [Olafsson 2002]. These

methods are not limited by the constraints of the input parameter space; i.e. the number of baggers available at a grocery store can only be a whole number.

Optimization methods vary depending on the complexity and type of configurable variables available for optimization. Gradient search methods, such as Response Surface Methodology and stochastic approximation, are commonly used for continuous variables. For finite and small systems ranking and selection methods may be used and for finite and large systems a metaheuristics should be applied [Olafsson 2002]. A metaheuristic is a non-analytical method of solving a problem that involves one heuristic driving a lower level heuristic or set of rules, to move throughout the search space.

2.2.5 Models of Simulation Models (meta-models)

These previous computational methods require a large amount of simulations in order to be effective which is why they are often used in deterministic optimization. In order to allow optimization techniques to work efficiently, a model of an initial output dataset is constructed. This is referred to as a meta-model (see Section 2.4 for more information) and it simplifies the simulation model itself, exposing the fundamental nature of the system input and output relationship [Santos, 1999]. This model can then be acted upon by the optimization method.

2.2.6 Response Surface Methodology

Response Surface Methodology (RSM) appears to be an obvious choice for creating a representative model for optimization. RSM consists of a collection of mathematical and

statistical methods utilized for experimental optimization as stated by Joshi *et al* [1998] in his paper describing an enhanced RSM algorithm. RSM incorporates gradient deflection methods and restart criteria as opposed to the traditional method of utilizing steepest ascent or descent direction. They describe the conventional method of RSM as initiating the described method by performing an experimental design in a small sub-region of the available space and modeling the data with a low degree polynomial. The method then proceeds to climb down the response surface. The typical RSM stops when the response either moves opposite to the desired direction or shows no further improvement. This method works under the assumption that the true optimum can be obtained on this path thus having very limited exploration. At this point, another experimental design is conducted to determine curvature and create a two degree polynomial analysis. A canonical analysis is then conducted to determine if further exploration needs to be conducted. Next a ridge analysis may be performed to collect observations along the ridge direction. From here the analysis may continue using discrete step sizes to determine "the optimal" in that region. The method relies on a random restart or multi-start technique to explore the experimental space [Joshi, 1998].

The *Enhanced RSM Algorithm* introduced by Josh et all (1998) initializes at a random point as well as treats this point as the incumbent solution. A 2K factorial experiment is constructed with the incumbent solution at the center. The path of steepest descent is selected as the search direction and a step length is derived on a line search so as to not

violate the bounds of the experiment. Following determination of the appropriate step length, a second order model is created. After the model is checked for adequacy, a canonical analysis is performed. Then the algorithm is repeated until the termination criteria are met. The can be either a set of predetermined criteria or no further improvement is commonly used [Joshi, 1998].

Neddermeijer *et al* (2000) created a framework for a fully automated RSM framework. A graphical representation can be seen in Figure 2.1. The framework initializes with a given starting point and step size. From the initial point, the response function is modeled using a first order model which is then tested for adequacy. Should the model not have a significant ability to predict the data, then the steepest descent direction cannot be determined. A line search is performed in the area of the steepest descent. The predetermined step size is taken in that direction. This is the optimum for the *n*th iteration of the framework. The line search is continued in that direction until the stopping criteria has been met. Other options that take the noise of the model into consideration stop when two or three consecutive points in that direction continue to achieve less desirable solutions. Should the first order model be found inadequate and there is evidence of pure curvature or interaction effects, a second order model can be applied to increase the accuracy of the model. To maintain the efficiency of the framework is has been found beneficial to reduce the region of interest thus limiting the search area.

Once the termination criterion has been met, the new region of interest is represented by a second order model which is also tested for adequacy. Again, if the model is found to be inadequate, the size of the new region of interest can be reduced by reducing the step size. Should the second order model be found to be adequate, a canonical analysis is performed to determine the nature and location of the current optimal solution. Then a ridge analysis is performed if the current optimal solution is found to be a saddle point [Neddermeijer 2000]. Ridge analysis computes the estimated ridge of optimum response using increasing radii from the original data point. The direction of steepest descent is determined for the second order model and a line search is performed in that direction. Once the optimum point in this new direction is found, the algorithm is repeated until an optimum solution is found [Neddermeijer 2000].

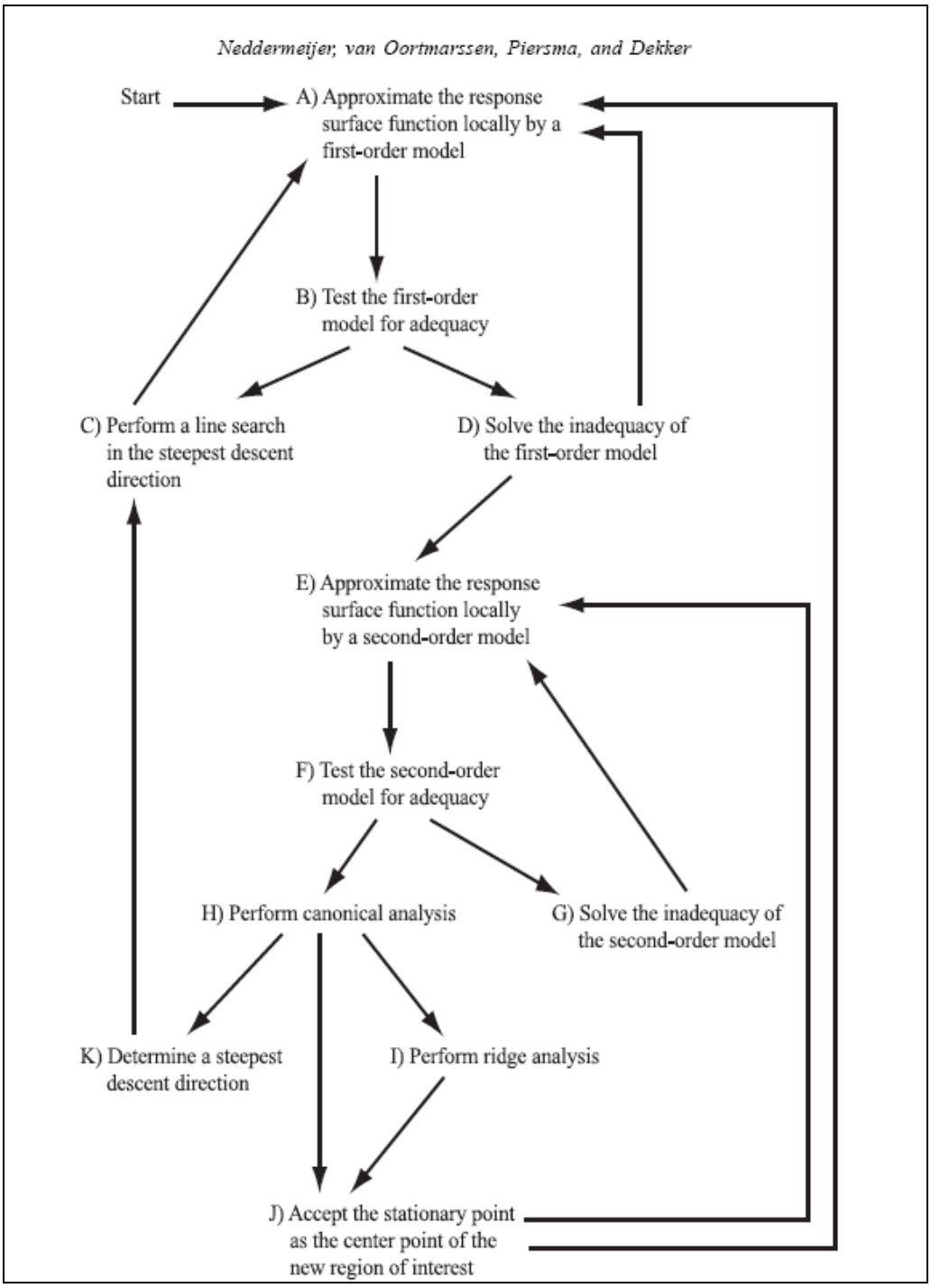


Figure 2.1: RSM Method (Taken from Neddermeijer et al (2000))

RSM provides a logical way to search for an optimal value for an experiment with variability. RSM relies on a computational optimization technique called random restart to fully explore the sample space. RSM depends on the linearity of the problem being optimized which presents a weakness in the method. RSM optimization depends on the ability to create both first and second order equations on the sample space as well as the ability to create differential equations from the derived models. This presents a problem in the goal of a completely automated program.

2.3 Genetic Algorithms

Complex problems like supply chain optimization are very difficult to solve. There is often no underlying analytical model and often a nonlinear relationship between the variable(s) being optimized and the prediction variables. This, coupled with the size and complexity of the search space, creates a seemingly impossible problem. Genetic algorithms have been shown to produce good results for a wide range of problems though the use of their powerful global search techniques based on Darwin's theories of survival of the fittest and random mutation. Genetic algorithms both explore the sample space and exploit promising areas exponentially. This is accomplished through mutation, crossover, and selection operations. Generally, the strongest individuals survive and reproduce from generation to generation.

This improves the overall strength of the population in successive generations and focuses on the most promising solution space. Due to the stochastic nature of the crossover and mutation procedures, weak individuals or children will be created. These individuals add to the explorative properties of the algorithm. They are not detrimental to the overall strength of the program since they are removed through the selection process during subsequent generations. If a weak solution should survive a generation, it will eventually be removed and thus not cause serious damage to the algorithm or hinder the final solution. The next generation of the population is created through random methods and evaluated by comparative selection based on a random underlying pairing. This differs from traditional optimization methods which are based on predetermined decision rules. Genetic algorithms do not make strong assumptions about the form of the objective function; thus allowing them to operate on a large range of problems [Joines 2002]. The general genetic algorithm is summarized in Figure 2.2.

1. Set generation counter $i \leftarrow 0$.
2. Create the initial population, P_0, by randomly generating N individuals.
3. Determine the fitness of each individual in the population by applying the objective function to the individual and recording the value found.
4. Increment to the next generation, $i \leftarrow i + 1$.
5. Create the new population, P_i, by selecting N individuals stochastically based on the fitness from the previous population, P_{i-1}.
    a. Randomly select R parents from the new population to form the new children by application of the genetic operators.
    b. Evaluate the fitness of the newly formed children by applying the objective function.
6. If $i <$ the maximum number of generations to be considered, go to Step 4.
7. Output the best solution found.

Figure 2.2: A Simple Genetic Algorithm

The solutions, or individuals, in the population are described by a chromosome representation consisting of a vector representing the predictive variables. The algorithm is initiated by creating an initial population and then evaluating their fitness to the objective function. A subset of the population is then selected to parent the next generation. It is possible for an individual to be selected to parent more than one offspring. Either crossover or mutation is performed to create the offspring. For example, one crossover consists of randomly selecting a cut point on the parents then swapping the numbers after the cut point between one another creating two distinct children. Crossover combines traits from both parents to create the new children. Mutation occurs when a random individual is selected and a random chromosome change is performed. The goal of mutation is to inject new information into the solution set, thus exploring different areas of the sample space. A subset of the old population is chosen to complete the new population. This subset can be chosen

either arbitrarily or by selection methods, such as ranking, binary selection, etc. [Michalewicz]. Genetic algorithms have successfully been used to solve a wide variety of very complex problems [Joines 2002].

2.3.1 Five Components of Genetic Algorithms

Genetic algorithms must consist of at least five components in order to operate [Michalewicz].

Chromosome Representation – A chromosome is the representation of the individual potential solution point. These *individuals* are typically represented in a binary method.

Initial Population – A method in which to create the initial population of potential solutions [Michalewicz]. In this case we will be pulling 100 data points from the training population for the meta-model.

Evaluation Function – This is a function that determines the rating of the solution mimicking the environment to select the "fittest" solutions [Michalewicz]. We will be using the meta-model of the neural network, regression model, actual evaluation function or a combination of them.

Genetic Operators – These alter the composition of the children of the population [Michalewicz]. These can be either crossover or mutation. *Crossover* is where two solution sets are combined to produce two children that are part of each solution, similar to mating. *Mutation* is the random changing of a part of the chromosome. This introduces randomness

to the solution set thus increasing the explorative nature of the algorithm. The frequency of these crossovers and mutations are parameters controlled within the genetic algorithm.

Termination Criteria – This determines how long the genetic algorithm will run. It can either be a specific number of generations or when the best solution does not change for a successive number of generations. The algorithm may also be terminated based on a number of evaluations of the evaluation function [Michalewicz]. In this research we will be using the number of iterations of the algorithm. The termination criteria will be set as a parameter in the experimental set-up.

Other parameters are entered into the algorithm such as the solution space boundary. This research utilized a solution space of [-150, 150] for each variable.

2.3.2 Research Algorithm

The genetic algorithm used in this research is modified from the *Binary and Real-Valued Simulation Evolution for Matlab* library created by C.R. Houck, J.A. Joines, and M.G. Kay in 1996.

2.4 Meta-model

A meta-model is an analytic approximation to a process with variability or the simulated response surface based on input variables to the simulation or real system and their resulting outputs. Utilizing a model provides a less computationally expensive and thus quicker method of evaluating the simulation or the real system. For this reason, optimization methods often use a meta-model for evaluation. Linear regression is one of the most common methods of developing a meta-model. Other methods are coming into use such as artificial neural networks, which are discussed below. The accuracy of this model and the frequency of which it is generated is a balancing act necessary to optimize performance while also optimizing accuracy.

2.4.1 Stepwise Regression

Regression is the most common means to create a meta-model. It creates a linear model by minimizing the sum of the squares of the residuals. The residuals are differences between the predicted values and the actual values. The least squares regression technique is based on the Least Squares Equations seen below in Figure 2.3

1. $\beta_1 = \dfrac{S_{xy}}{S_{xx}}$ where $S_{xy} = \sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$ and $S_{xx} = \sum_{i-1}^{n}(x_i - \overline{x})^2$

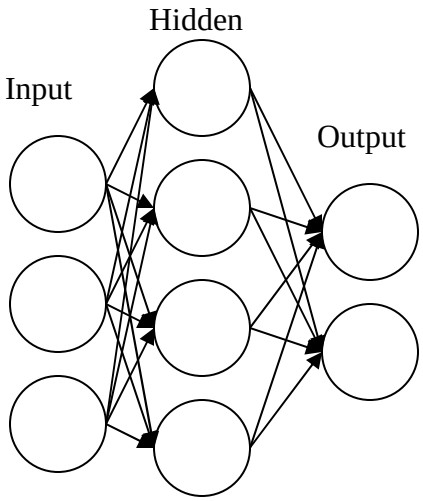
2. $\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 \overline{x}$

Figure 2.3: Least Squares Equations for Linear Regression Model (Wackerly 2002)

Stepwise regression is the method used to add or subtract the terms of a model in an attempt to better explain or predict the output by creating a formula based on the input variables and/or terms. This can be done using 1st, 2nd, 3rd, or even higher order inputs based on the desired complexity of the model. In forward stepwise regression, terms are added based on their ability to positively impact the model's prediction capabilities which is known as the significance, or *p*-value, of the term. The *p*-value is relevant to the amount of variability attributed to that term, thus, the closer to zero, the better. Following a term being added, the model is re-evaluated along with other potential terms to determine if any additional terms should be added. When no further terms meet the criteria for addition to the model, the process is terminated and that model is used as the meta-model to predict outputs based on predetermined inputs. [Rawlings 1998].

2.4.2 Neural Network

An Artificial Neural Network, commonly referred to as a neural network, is another method that can be used to create a meta-model and is often referred a nonlinear regression technique based on simulating biological neural networks. A neural network consists of nodes (neurons); input, hidden, and output nodes. These highly interconnected nodes work together to solve specific problems, see Figure 2.4: Basic Neural Network



.

Figure 2.4: Basic Neural Network

Similar to biological neural networks, artificial neural networks must be trained to solve the problem they are modeling. There are three basic types of neural networks based on the learning paradigms: Supervised, Unsupervised, and Reinforcement. Supervised training uses input and corresponding output data to train the model to predict outputs from new unseen inputs which is basically a regression technique. Therefore, the remainder of the discussion will focus on supervised learning neural networks. In order to accomplish the modeling of a simulation, it is necessary to have a training population based off the simulation that can be used as a starting point to train the neural network. The accuracy of the meta-model created in the neural network is dependent on the amount of data in the training population.

2.4.3 Neural Network Training

There are multiple ways to train a neural network. A widely used method is backpropagation. This is a gradient descent method based on minimizing the error between the actual output of the training population and the projected output of the neural network which is similar to the output of least squares regression method. There are two methods that can be used to train the neural network where training refers to setting the weights of the various nodes. The first method introduces one training set (i.e., one set of input and output combinations) at a time which is referred to as the pattern form, while the second introduces the entire training population or batch form (Li 2001). The backpropagation algorithm of both methods is similar in that weights are adjusted based off the network predicted outcome and the training set outcome.

Backpropagation Algorithm (Li 2000):

1. The training population is presented to the neural network which has random weights assigned.
2. The input variables are passed through the network and the network output (y_p) is compared against the actual output (y_a) from the training set. The error is calculated by:

$$\varepsilon(n) = \sum (Ya - Yp)^2 \quad \text{– Batch Training}$$
$$\varepsilon(n) = 1/2(d(n) - y(n))^{\tau}(d(n) - y(n)) \quad \text{- Pattern Form}$$

3. Adjust the weights of the each node based on the gradient and calculated error to minimize the difference in step 2
4. Assign weights to previous layers in the same manner.
5. Repeat until the error is acceptable (<1% of the output) or the termination criteria is met (Roja 1996).

In the pattern form of backpropagation the error is calculated on an individual training point and the weights adjusted in small increments. This is continued for each of the training points in the training population. One cycle of the complete training population is called an epoch. Following completion of this epoch, the data points are assigned random numbers and re-fed through the algorithm. This is completed until a termination criterion is reached (Li 2000). The batch form performs in a similar manner except that the error of a set of training points is computed and the weights adjusted. The training batch does not necessarily make up the entire training population (Li 2000).

There are many variations of the backpropagation algorithm. The simplest is the one where the weights are adjusted in the direction that causes the greatest impact on the error function, *Gradient Descent*. The amount that the weights are adjusted is constant. A variation of this method is to increase the amount of weight change each iteration in the same direction as long as the error function is being minimized, or a *Learning Gradient Descent.* *Gradient Descent Momentum* allows the network to respond to the recent trends in the gradient surface as well as the most recent gradient. This prevents the training algorithm from getting hung-up in a local minimum [Matlab- 2009].

Other methods exist to further speed up the network training and expedite convergence. The network used in the forthcoming experiments utilizes a scaled conjugate gradient algorithm developed by Bishop in 1995 in his paper *Network for Pattern Recognition*. The *Conjugate Gradient Search* searches along the conjugate gradients for the

steepest descent path as opposed to only the steepest descent single gradient path [Matlab - 2009]. One the network has been trained it can then be used as a meta-model for the optimization function to operate on.

The neural network used in this paper is modified from the netlab library for Matlab based of the approach and techniques described in *Neural Networks for Pattern Recognition* (Bishop 1995). The network itself consists of two layers; comprised of one input layer and one hidden layer, feed forward network with a single output. The hidden layer consists of ten hidden nodes with the input layer having the same number of input nodes as input variables. Node weights prior to training are randomly assigned from the 0 mean Gaussian distribution. The size of the training population and the termination criteria is an experimental criterion.

2.5 Simulation Optimization – Why we need meta-models

Simulation is computationally a very expensive operation. The computational cost varies based on the complexity of the simulation itself. Stochastic inputs will further increase this cost due to the necessity to run multiple simulations for each input variable setting to determine the average value of the output or confidence. That being said, optimization methods used on the simulation itself can be extremely expensive. An optimization method such as RSM would involve multiple simulations for each search point. A genetic algorithm

requires a large population and utilizes the computer to generate many points during each generation. Many individuals are often poor and will die off in subsequent generations. Also early in the search, the GA just needs to move into the general area of the optimal solution before it fine tunes to find the actual optimal. Each new individual requires an evaluation which in this case is a simulation model. It is apparent how the cost of these optimization methods would be unacceptable. Therefore, it is highly desirable to utilize a less expensive surrogate model (i.e., meta-model). The optimization function can use the meta-model for evaluation for a portion of the time and as well as early in the cycle to reduce the amount of true simulations needed. The goal will be to combine a simulation meta-model with the optimization tool to create a more efficient and effective optimization technique. The goal is to determine the best settings when using a RSM and Neural Networks as the meta-models to assist in the determination of the optimum value in a simulation.

2.6 Approach

Specific optimization problems were chosen to represent simulation models and have been shown to cause difficulty for deterministic optimization methods. A genetic algorithm will be used for the optimization technique and neural networks and step-wise regression for two potential meta-modeling techniques.

In Chapter Three, the first step will be to determine the effectiveness of each meta-modeling technique and the effectiveness of a number of variables on that effectiveness. In

order to evaluate this, we will conduct the experiment on two non-linear problems with known solutions, the Brown and Schwefel problems. Initially, the two options will be tested in a full-factorial design in a deterministic experiment. This experiment will allow the meta-modeling techniques and optimization algorithm to operate with the distraction of noise. Different attributes of each meta-modeling technique will be tested at various levels to determine the effect, if any, that they have on the effectiveness of the meta-model in determining the optimal solution. Following this experiment noise will be added to the problem in Chapter four. This is designed to test the meta-model's performance in a stochastic environment which is similar to real world circumstances.

3 Deterministic Experimental Procedure

Most simulations are stochastic in nature and therefore produce stochastic output as stated in Chapter 2. This additional level of complexity (i.e., noisy solution values) may confound other factors. Therefore the first experiment will eliminate the noise factor and solve several known deterministic optimization problems utilizing meta-models which will serve two purposes. First it will determine which factors (number of training points, response surface methodology, etc) provide the most influence on the quality of the final solution, where the quality of the final solution is determined by its overall value and computational efficiency. Second, the experiment will also determine the effectiveness of linear regression and neural networks in successfully modeling the complex functions.

The first experiment will minimize two sample problems under a different number of variables, the Schwefel function and Brown's almost linear function [Humphrey 2000]. The two variable surface plots are exhibited in Figure 3.2 and Figure 3.3.

Schwefel (Sine Root) Function

$$\theta(X) = -(122.8762 * n + sum\sum(-x * \sin(\sqrt{|x|}) - 1)$$

Brown's Almost Linear Function

$$\theta(X) = -\sum_{t=1}^{d}[f_{f1}(x)]^2 + 1$$

Where

$$f_i(x) = x_i + \sum_{j=1}^{d} x_j - (d+1) \quad \text{for}$$

i = 1, …, d-1, and

$$f_d(x) - (\prod_{j=1}^{d} x_j) - 1$$

Optimal Values

Y= -1

Y= -1

Figure 3.1: Schwefel and Brown Function Equations and Optimal Values

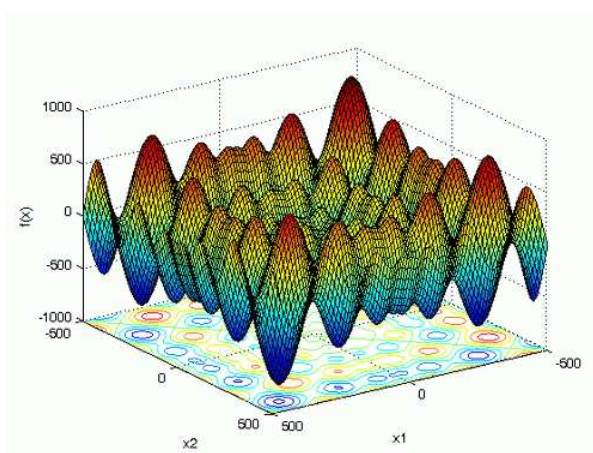


Figure 3.2: Schwefel Response Surface for nv=2 [Iframe]

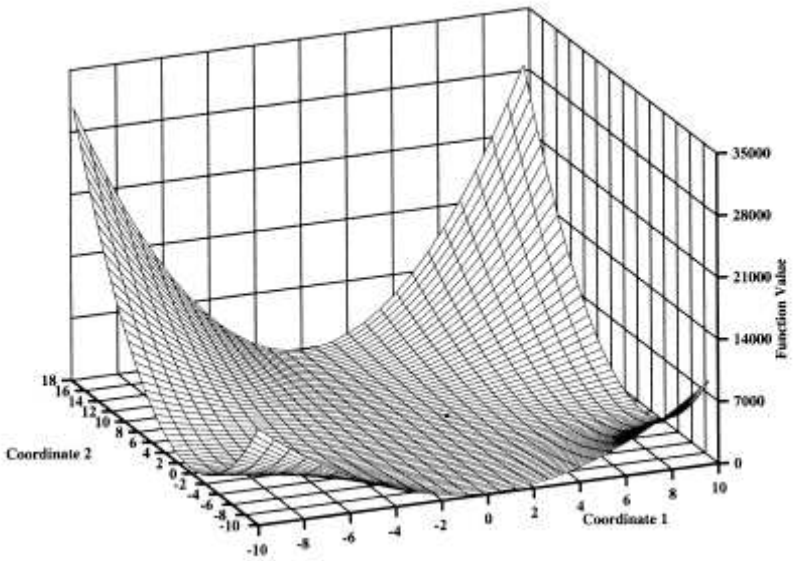


Figure 3.3: Brown's Almost Linear Function [Humphrey, 2000]

As mentioned in Chapter 2, genetic algorithms typically maximize functions. Therefore, these minimization functions are turned into maximization problems by negating the solution value, in effect, mirroring them around the *X*-axis. The sample functions were designed to represent the non-linear nature and complexity of an actual real world problem. The initial experiment also serves to determine whether or not the linear regression modeling technique is capable of approximating non-linear models. The initial suspicion is that it will not yield a high degree of approximation; however it may yield a capable method of removing some of the load off of the true evaluation functions.

3.1 Experimental Setup

The ultimate goal is to develop an effective simulation optimization methodology. In order to develop simulation optimization methodologies, researchers often imitate a true simulation by adding noise to a known deterministic optimization problem. Therefore, the evaluation function only takes seconds rather than minutes or hours to evaluate. The noise is frequently nothing more than a random number generated from a known distribution, such as the normal, with a known standard deviation. Humphrey and Wilson (2000) have shown this technique to be a capable method of simulation optimization modeling. Again, the noise term is omitted in the first experiment to create a deterministic evaluation function. A deterministic evaluation function allows for a clear understanding of the effect the parameters being tested have on explaining the variation within the experiment and their significance.

Four parameters are tested that are applicable to both the neural network and stepwise regression meta-modeling techniques. An additional parameter that is uniquely applicable to the stepwise regression meta-modeling technique is also evaluated within the stepwise regression portion of the experiment. This parameter is the significance factor that is achieved for a term to be added into the model, or the ρ-value required. The four parameters for both modeling techniques are the size of the training population, the number of generations the genetic algorithm searches, the amount of time the true evaluation function is used to evaluate the generated data point, and a retraining parameter. These parameters are explained in further detail in Section 3.1.1.

To fully understand the effect and significance that a parameter may have on the meta-modeling technique, it is necessary to evaluate that parameter on problems of various complexities. Therefore, the experiment has three different variable settings for each of the two optimization functions representing various levels of complexity. A function with two variables represents a simpler problem while ten and twenty variable functions represent moderate to complex problems respectively.

In order maintain comparability of the different parameter values it was necessary to begin the genetic algorithm with the same starting population for each combination of parameter settings. This is accomplished by beginning the random number generator at a known value, or seed which ensures that the starting population will be same for each experimental run. Also, the common random number method is carried over to the replication portion of the experiment with each replication beginning at a known seed. For example, replication one begins at seed one, replication two at seed two, and so forth. Common random numbers will be particularly useful while testing the retraining parameter.

3.1.1 Experimental Parameters

The following parameters will be evaluated on their impact on the effectiveness of the various simulation optimization methodologies during this experimentation.

***NumVar*--**The number of variables used to obtain the output value. This controls the relative complexity of the problem as discussed above.

NumTrain--This represents the size of the training population. Both modeling techniques require a population of known output values to train the meta-model. The training is done by two different methods (i.e., Neural-Network back-propagation algorithm and least squares linear regression) but both work by reducing the total amount of error between the predicted value and the actual value as discussed. These actual values and predictive variables are taken from the training population. One goal is to determine whether the size of the training population has an effect on the final solution quality, and if so, what size provides the best solution quality.

TermOpts—The maximum amount of generations/iterations the genetic algorithm is allowed to search for the optimal solution. The meta-model is less complex in comparison to the actual function that it is modeling. Theoretically, this simpler version should be easier to solve taking the genetic algorithm less time to find the best solution. It is also an assumption that increasing the complexity of the problems by increasing the number of variables will require a larger number of generations to obtain a solution close to optimal.

TrueEval(*evalOpts*)--The percentage of times that the true evaluation function (i.e., the actual simulation) is used to evaluate the individuals created by the genetic algorithm as compared to the meta-model only. The goal of this parameter is to determine if evaluating the true function a portion of the time affects the solution quality since the meta-model is an approximation to the true model. It may be beneficial to access the true model periodically to validate the values produced by the meta-model. In order to determine this point in the evaluation function, a random number is generated and compared to the parameter value. If it is less, then the true simulations is used to evaluate the individual otherwise the meta-model is evaluated

Retrain--Retraining or not retraining between replications. The experiment is set up to utilize the same meta-model for as long as possible (i.e. that the meta-model is not retrained between replications, true evaluation parameter changes, and termination operator parameter changes). The various parameter settings utilize the same meta-model to obtain the solution. If retraining is used, the meta-model is retrained between every replication, creating a new meta-model for every time. Otherwise the same meta-model is used for all replications. This is used to determine if retraining the meta-model on a constant basis is beneficial to obtaining a high solution quality. The goal is to determine how beneficial is when taking into account to the computational effort needed to train a meta-model on a complex data set; one with many variables.

P-value--Represents the level of significance necessary for an explanatory variable to be added to the meta-model. This parameter is exclusive to the stepwise regression meta-modeling technique. If the probability that the effect of the explanatory variable is created by noise is less than the *p*-value dictated in the parameter, seen in Table 3.1: Parameters and Settings, then the explanatory variable is added to the meta-model. In turn, this acts as a method of excluding insignificant explanatory variables. Two parameter levels were used. The higher level will accept more terms into the meta-model, creating a more complex meta-model while also increasing the probability that an insignificant explanatory variable will be added to the model. The goal is to determine if this parameter has an effect on the final solution quality. Table 3.1 depicts the various parameters that will be used in this experiment along with the number of levels and their values.

Table 3.1: Parameters and Settings

Parameter	**Levels**				
NumVar	2	10	20		
NumTrain	200	500	1000		
TermOpts	100	300	600		
evalOpts	0	0.10	0.25	0.50	1.00
Retrain	0 - No	1 - Yes			
P-value	0.05	0.15			

3.1.2 Experimental Procedure Step-by-Step

The computational experiment can be subdivided into two basic procedures. The first procedure, depicted in Figure 3.4, deals with the experiment where the meta-model is not retrained between replications while Figure 3.5 shows the procedure where the meta-model is retrained after each replication.

1. Choose evaluation function – Schwefel or Brown
2. Choose number of variables – 2, 10, or 20
3. Set random number generator to random seed #Replication
4. Choose training sample size and create training population using chosen evaluation function
5. Fit meta-model utilizing chosen meta-modeling technique
6. Utilize meta model to find optimum value using different values of maximum number of generations and the amount of time the true function is used for evaluation
7. Repeat step 6 for nine additional search replications. Begin each search replication from a known random number generator seed
8. Repeat steps 1 thru 7 for each combination of function, number of variables, and size of the training population

Figure 3.4: Step-by-Step Procedure: No Retrain

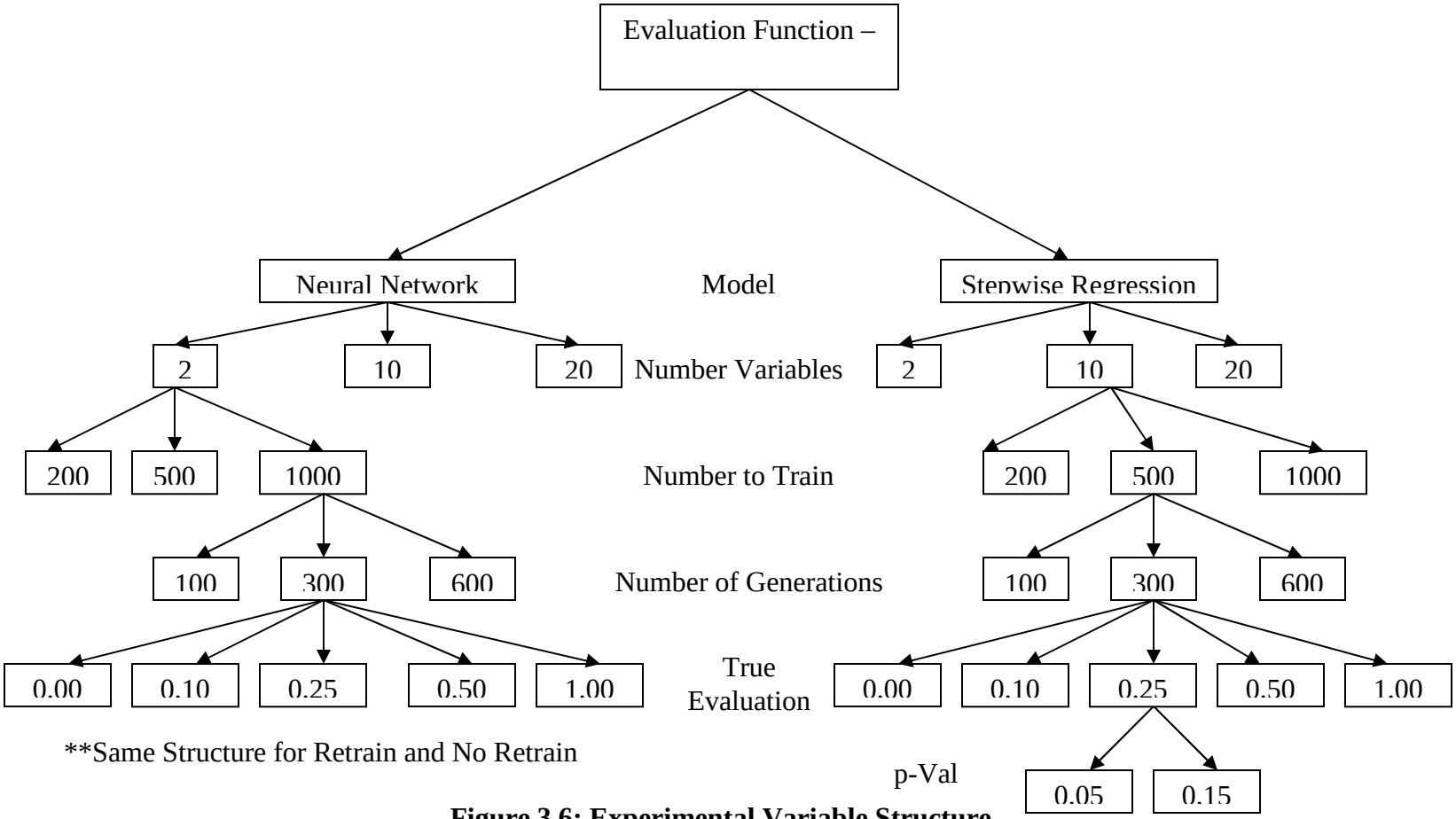
1. Choose evaluation function – Schwefel or Brown
2. Choose number of variables – 2, 10, 20
3. Choose training sample size
4. Choose maximum number of generations and evaluation options.
5. Set random number generator to known value corresponding with searching replication number.
6. Create training sample and fit meta-model using chosen meta-modeling technique
7. Optimize using genetic algorithm
8. Repeat the step 6 for nine additional search replications. Begin each search replication from a know random number generator seed
9. Repeat steps 5 thru 7 for all combinations of variables

Figure 3.5: Step-by-Step Procedure: Retraining Between Search Replications

3.2 Experimental Design

The experimental design which utilizes the procedures in Figure 3.4 and Figure 3.5 is a five factor, full factorial conducted on three different variable settings, two meta-modeling techniques, and two known functions. In effect, the experiment becomes an eight factor design. Only interaction effects for the initial five factors are observed: the number of variables, number to train, number of generations, true evaluation ratio, and ρ-value. A tree representation of the experimental design can be seen below in Figure 3.6.

Deterministic Experiment Setup



**Same Structure for Retrain and No Retrain

Figure 3.6: Experimental Variable Structure

3.3 Output Data

The experimentation code is set up to record the parameter values along with a series of other values that will determine the efficiency and the effectiveness of the combined meta-modeling and search technique. The number of times that the meta-model is evaluated is kept along with the number of times the true evaluation function is used to determine if there is a relationship between those values and the final outcome. When the genetic algorithm finds the best solution, the generation is stored to determine the efficiency of the search technique relative to the parameter values in the experiment.

3.4 Results

Following the conclusion of the experiment, the data is collected and analyzed to determine the significance of the parameter effects. The analysis is divided by problems, Schwefel and Brown, and then further divided by meta-modeling techniques. The significance values in Table 3.2 represent the probability that the parameter effect is caused by noise. The significance test is conducted on the maximum number of generations, the size of the training population, and the amount of time to evaluate the true model represented as both a continuous and a categorical variable to determine the effect that varying the modeling technique poses on the significance level. The change in modeling type resulted in different significance values as expected. The significant parameters in the neural network meta-

modeling technique did not change drastically. However, the modeling type change in the regression meta-modeling technique created a significant difference in the significant terms as seen in Table 3.6 . This shift is discussed further in Regression. Using the screening test results from this chapter, the total number of variables can be reduced and only the terms that are determined important have to be screed in the full model of chapter five. This produces a reduced number of parameters that has to be statistically analyzed and compared.

3.4.1 Neural Network

Analysis of the neural network meta-modeling technique provides insight into the parameters that hold the most significance on the final solution quality. The continuous and categorical variable parameter modeling types produce very similar parameter significances. The exception is the variable associated with the maximum number of generations, amount of time the true model is evaluated, and the retraining parameter interaction as seen in Table 3.2. Due to the experimental setup it is necessary to separate the analysis into retrain and non retrain sections. This is due to the fact that when retraining is not used the effect of the size of the training population is not clear. It is not clear because the meta-model is only recreated when adjusting the size of the training population. The significant effects of the retraining portion of the experiment are presented in using the parameter names explained in section 3.1.1. Significance levels high-lighted in yellow represent the parameters that are found to be statistically significant. A Residual Maximum Likelihood Estimation (REML) was utilized to determine the significance of the terms as it takes into account the loss of

degrees of freedom with the analysis. A random parameter, the repetition number, was added to the analysis (***rep&random***) and a random interaction (***rep&random*numTrain***) to test the significance evaluation.

Table 3.2: Neural Network Parameter significances (significant parameters high-lighted in yellow)

			Schwefel				Brown		
	DF	**DFDen**	**2**	**10**	**20**		**2**	**10**	**20***
TermOpts	2	2	0.876	0.453	0.017		0.769	0.862	0.000
numTrain	2	2	0.133	0.000	0.000		0.000	0.097	0.000
TermOpts*numTrain	4	4	0.828	0.964	0.711		0.216	0.851	0.000
evalOpts	4	4	0.000	0.000	0.000		0.000	0.077	0.000
TermOpts*evalOpts	8	8	0.950	0.725	0.508		0.922	0.896	0.000
numTrain*evalOpts	8	8	0.008	0.000	0.000		0.010	0.196	0.000
TermOpts*numTrain*evalOpts	16	16	0.993	0.501	0.896		0.938	0.940	0.000
Retrain	1	1	0.000	0.000	0.047		0.000	0.030	0.000
TermOpts*Retrain	2	2	0.877	0.678	0.565		0.762	0.859	0.000
numTrain*Retrain	2	2	0.000	0.000	0.000		0.000	0.038	0.000
TermOpts*numTrain*Retrain	4	4	0.827	0.869	0.963		0.210	0.852	0.000
evalOpts*Retrain	4	4	0.000	0.177	0.000		0.000	0.594	0.000
TermOpts*evalOpts*Retrain	8	8	0.950	0.654	0.476		0.921	0.895	0.000
numTrain*evalOpts*Retrain	8	8	0.000	0.000	0.023		0.003	0.340	0.000
TermOpts*numTrain*evalOpts*Retrain	16	16	0.993	0.456	0.938		0.938	0.940	0.000

*Due to the complexity of the Brown non-linear problem with 20 variables the neural-network was unable to effectively model the problem.

The retraining data set is utilized to determine the significant effects due to the previously mentioned reasons. The analysis inTable 3.2: Neural Network Parameter significances (significant parameters high-lighted in yellow) shows that the size of the training population, the amount of time that the true function is evaluated and the interaction effects are the most significant in the Schwefel function, Table 3.2 also shoes that the two and ten variable

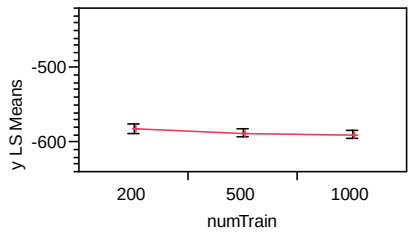
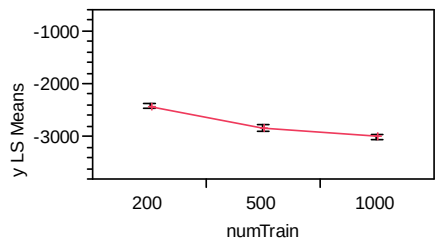
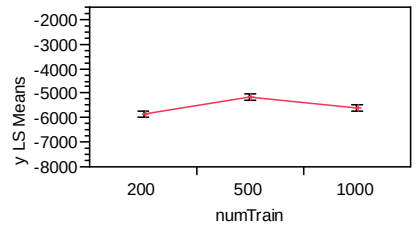
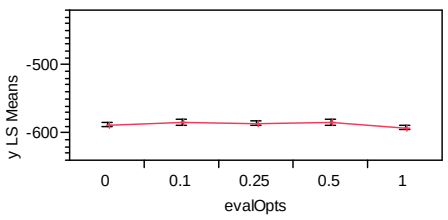
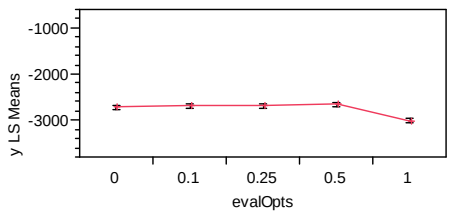
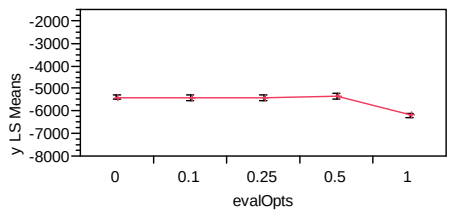
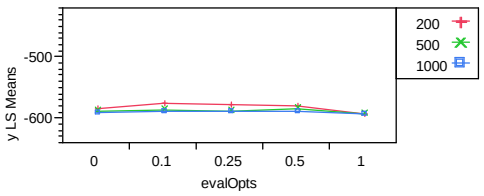
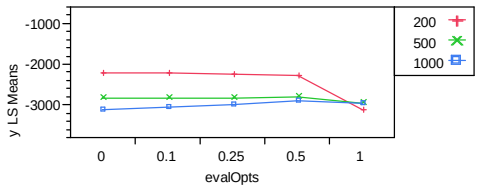
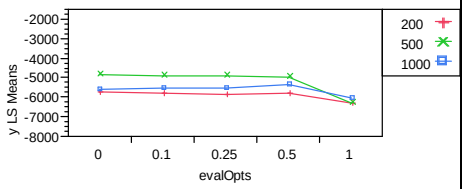
options for both retraining and not retraining the Brown function. The effect tests presented in Figure 4.2 show that a higher training population size produces a value closer to optimal. The larger the amount of time that the true function is utilized, the closer a solution to optimal that is found. The invalid *p*-values associated with the significance test on the 20 variable case of Brown's function are caused by the large amount of variance in the true values. This is caused by the inability of the neural network to estimate the function.

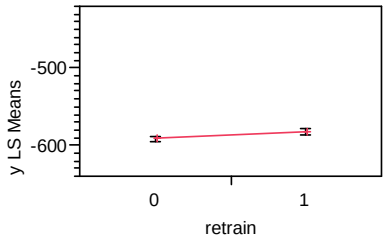
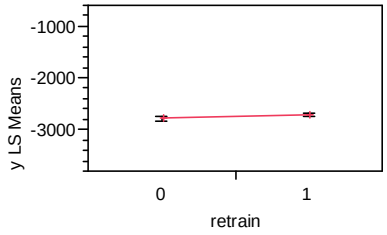
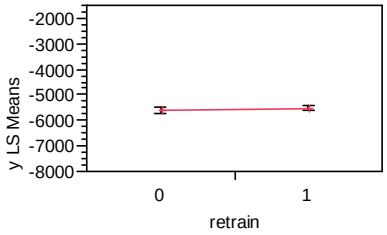
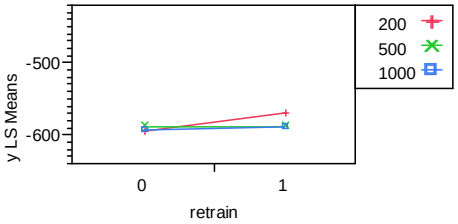
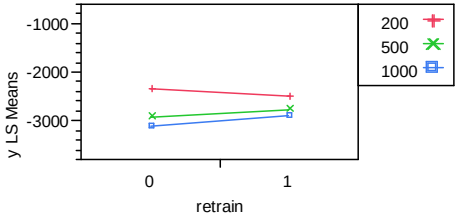
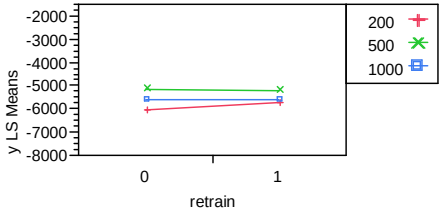
The analysis will focus on the parameters and 2nd order interactions that are most prevalent between the two problems. These are: ***numTrain***, ***evalOpts***, ***numTrain*evalOpts***, ***retrain***, ***numTrain*retrain*** and ***evalOpts*retrain***. The 3rd order and higher interactions will be ignored owing to the simplicity of this experiment. The 3rd order interaction of ***numTrain,*evalOpts*retrain*** is statistically significant but that can be attributed to the interactions between the three significant factors that make up the interaction. Figure 3.7and Figure 3.8 show the effect plots of the parameters mentioned previously. The least squared means analysis are included below in Table 3.3 and Table 3.4 respectively for both the Schwefel and Brown functions. The 20 variable Brown function was omitted due to the inability of the neural network to model the complex function.

The training population size, ***numTrain***, is a significant parameter but does not show any sign of correlation between the size of the training population, the complexity of the problem and the effectiveness of the algorithm to find the optimal solution. The training size shows no significance at the lower complexity Schwefel problem and has an optimal value of

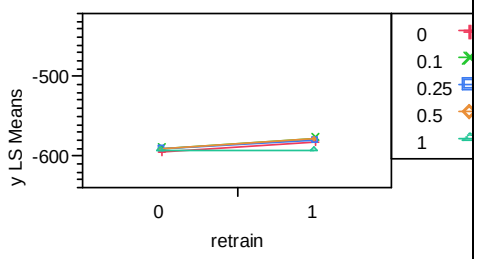
1000 at the 10 variable and 500 at the 20 variable problem. On the more complex Brown problem, the higher training population size allows the algorithm to find a better solution. The standard error of the ten variable Brown function is so large that one cannot tell the difference between the three parameter settings.

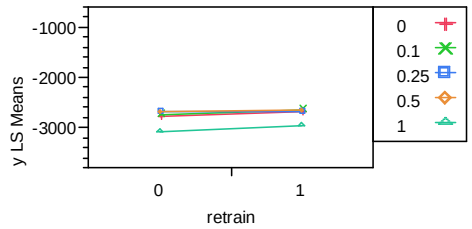
Figure 3.7: Schwefel Effect Analysis

	Schwefel -2	Schwefel – 10	Schwefel – 20
numTrain			
evalOpts			
numTrain*evalOpts			

	Schwefel -2	Schwefel – 10	Schwefel – 20
retrain			
numTrain*retrain			

evalOpts*retrain





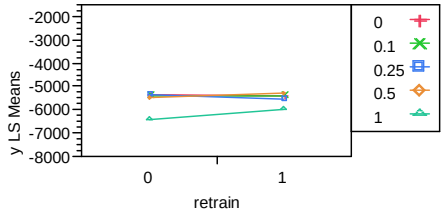


Table 3.3: Least Sq Means Table – Schwefel Function

	Schwefel – 2			Schwefel – 10			Schwefel - 20		
numTrain	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**
	200	-582.4099	2.6863683	200	-2418.413	28.363880	200	-5878.196	58.711612
	500	-588.2419	2.6863683	500	-2848.588	28.363880	500	-5176.750	58.711612
	1000	-590.2312	2.6863683	1000	-3004.070	28.363880	1000	-5621.916	58.711612
evalOpts	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**
	0	-588.1223	1.7835864	0	-2725.433	21.850429	0	-5381.560	59.347293
	0.1	-584.2289	1.7835864	0.1	-2694.474	21.850429	0.1	-5419.975	59.347293
	0.25	-585.5971	1.7835864	0.25	-2689.984	21.850429	0.25	-5422.498	59.347293
	0.5	-584.7163	1.7835864	0.5	-2658.644	21.850429	0.5	-5372.348	59.347293
	1	-592.1406	1.7835864	1	-3016.581	21.850429	1	-6198.388	59.347293

Table 3.3: Continued

	Level	Least Sq Mean	Std Error	Level	Least Sq Mean	Std Error	Level	Least Sq Mean	Std Error
*numTrain*evalOpts*	200,0	-584.4144	3.1270712	200,0	-2228.585	41.836455	200,0	-5698.388	96.540786
	200,0.1	-576.2259	3.1270712	200,0.1	-2201.896	41.836455	200,0.1	-5815.774	96.540786
	200,0.25	-578.7461	3.1270712	200,0.25	-2245.990	41.836455	200,0.25	-5835.804	96.540786
	200,0.5	-580.4499	3.1270712	200,0.5	-2292.643	41.836455	200,0.5	-5775.429	96.540786
	200,1	-592.2134	3.1270712	200,1	-3122.950	41.836455	200,1	-6265.586	96.540786
	500,0	-589.6444	3.1270712	500,0	-2827.637	41.836455	500,0	-4829.533	96.540786
	500,0.1	-586.6673	3.1270712	500,0.1	-2835.728	41.836455	500,0.1	-4918.911	96.540786
	500,0.25	-588.4530	3.1270712	500,0.25	-2825.028	41.836455	500,0.25	-4888.855	96.540786
	500,0.5	-584.4498	3.1270712	500,0.5	-2790.713	41.836455	500,0.5	-4962.140	96.540786
	500,1	-591.9949	3.1270712	500,1	-2963.831	41.836455	500,1	-6284.314	96.540786
	1000,0	-590.3080	3.1270712	1000,0	-3120.078	41.836455	1000,0	-5616.760	96.540786
	1000,0.1	-589.7934	3.1270712	1000,0.1	-3045.798	41.836455	1000,0.1	-5525.240	96.540786
	1000,0.25	-589.5921	3.1270712	1000,0.25	-2998.935	41.836455	1000,0.25	-5542.836	96.540786
	1000,0.5	-589.2493	3.1270712	1000,0.5	-2892.574	41.836455	1000,0.5	-5379.476	96.540786
	1000,1	-592.2134	3.1270712	1000,1	-2962.963	41.836455	1000,1	-6045.265	96.540786
	Level	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**
retrain	0	-591.6710	1.5939513	0	-2789.065	15.524060	0	-5602.912	45.319242
	1	-582.2510	1.5939513	1	-2724.981	15.524060	1	-5514.996	45.319242

Table 3.3: Continued

	Level	Least Sq Mean	Std Error	Level	Least Sq Mean	Std Error	Level	Least Sq Mean	Std Error
numTrain*retrain	200,0	-594.5393	2.8030474	200,0	-2328.215	32.263826	200,0	-6044.386	70.109349
	200,1	-570.2806	2.8030474	200,1	-2508.611	32.263826	200,1	-5712.006	70.109349
	500,0	-588.1835	2.8030474	500,0	-2933.804	32.263826	500,0	-5138.380	70.109349
	500,1	-588.3003	2.8030474	500,1	-2763.371	32.263826	500,1	-5215.120	70.109349
	1000,0	-592.2902	2.8030474	1000,0	-3105.178	32.263826	1000,0	-5625.971	70.109349
	1000,1	-588.1723	2.8030474	1000,1	-2902.961	32.263826	1000,1	-5617.861	70.109349
	Level	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**
evalOpts*retrain	0,0	-594.5185	2.0612327	0,0	-2781.641	29.521449	0,0	-5354.392	77.260687
	0,1	-581.7260	2.0612327	0,1	-2669.225	29.521449	0,1	-5408.729	77.260687
	0.1,0	-590.6168	2.0612327	0.1,0	-2732.786	29.521449	0.1,0	-5441.721	77.260687
	0.1,1	-577.8409	2.0612327	0.1,1	-2656.162	29.521449	0.1,1	-5398.229	77.260687
	0.25,0	-590.5811	2.0612327	0.25,0	-2690.408	29.521449	0.25,0	-5337.211	77.260687
	0.25,1	-580.6130	2.0612327	0.25,1	-2689.561	29.521449	0.25,1	-5507.786	77.260687
	0.5,0	-590.5707	2.0612327	0.5,0	-2670.447	29.521449	0.5,0	-5450.028	77.260687
	0.5,1	-578.8619	2.0612327	0.5,1	-2646.840	29.521449	0.5,1	-5294.669	77.260687
	1,0	-592.0677	2.0612327	1,0	-3070.045	29.521449	1,0	-6431.210	77.260687
	1,1	-592.2134	2.0612327	1,1	-2963.118	29.521449	1,1	-5965.566	77.260687

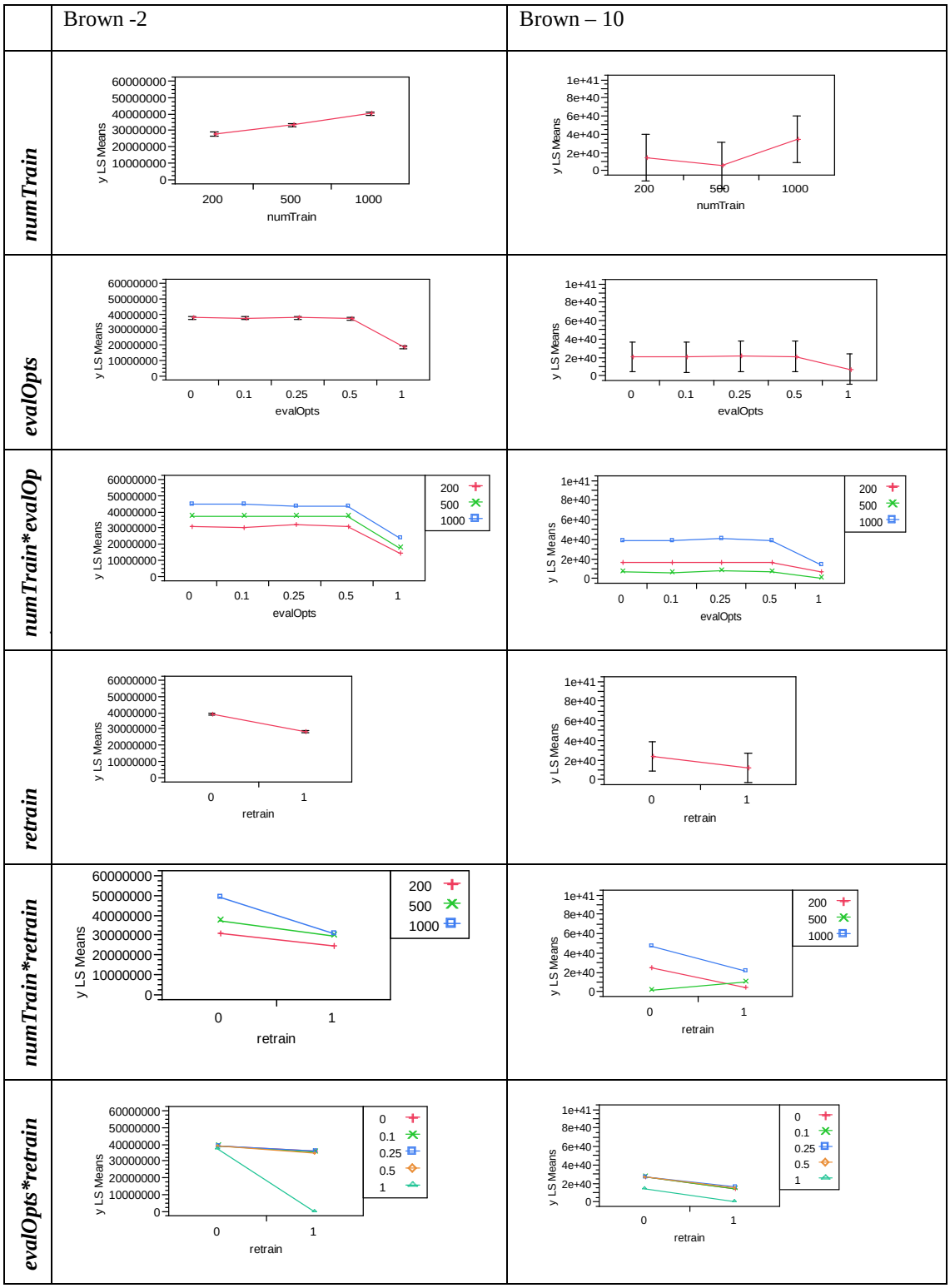


Figure 3.8: Brown Effect Plots

Table 3.4: Brown Least Sq Means

	Brown – 2			Brown – 10		
numTrain	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**
	200	27739813	507627.18	200	1.4551e+40	2.0272e+39
	500	33306910	507627.18	500	5.4902e+39	2.0272e+39
	1000	40126983	507947.14	1000	3.4203e+40	2.0272e+39
evalOpts	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**
	0	37660519	460429.16	0	2.0625e+40	1.2985e+39
	0.1	37431941	459447.90	0.1	2.0369e+40	1.2985e+39
	0.25	37574632	459447.90	0.25	2.1623e+40	1.2985e+39
	0.5	37228954	459447.90	0.5	2.0869e+40	1.2985e+39
	1	18726796	459447.90	1	6.9204e+39	1.2985e+39
numTrain*evalOpts	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**
	200,0	30926574	774424.02	200,0	1.6581e+40	2.264e+39
	200,0.1	30371506	774424.02	200,0.1	1.6768e+40	2.264e+39
	200,0.25	32025912	774424.02	200,0.25	1.6641e+40	2.264e+39
	200,0.5	30800422	774424.02	200,0.5	1.6527e+40	2.264e+39
	200,1	14574651	774424.02	200,1	6.239e+39	2.264e+39
	500,0	37539275	774424.02	500,0	7.0195e+39	2.264e+39
	500,0.1	37218298	774424.02	500,0.1	5.9738e+39	2.264e+39
	500,0.25	37037995	774424.02	500,0.25	7.2903e+39	2.264e+39
	500,0.5	37211517	774424.02	500,0.5	6.9227e+39	2.264e+39
	500,1	17527464	774424.02	500,1	2.4452e+38	2.264e+39
	1000,0	44515710	779651.42	1000,0	3.8273e+40	2.264e+39
	1000,0.1	44706020	774424.02	1000,0.1	3.8365e+40	2.264e+39
	1000,0.25	43659990	774424.02	1000,0.25	4.0938e+40	2.264e+39
	1000,0.5	43674923	774424.02	1000,0.5	3.9159e+40	2.264e+39
	1000,1	24078273	774424.02	1000,1	1.4278e+40	2.264e+39
retrain	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**
	0	39033115	354577.20	0	2.4074e+40	1.1967e+39
	1	28416022	354373.48	1	1.2089e+40	1.1967e+39
numTrain*retrain	**Level**	**Least Sq Mean**	**Std Error**	**Level**	**Least Sq Mean**	**Std Error**
	200,0	30764874	585830.31	200,0	2.4423e+40	2.0889e+39
	200,1	24714752	585830.31	200,1	4.6793e+39	2.0889e+39
	500,0	37116785	585830.31	500,0	8.9086e+38	2.0889e+39
	500,1	29497035	585830.31	500,1	1.0089e+40	2.0889e+39
	1000,0	49217687	586938.63	1000,0	4.6907e+40	2.0889e+39
	1000,1	31036279	585830.31	1000,1	2.1498e+40	2.0889e+39

Table 3.4: Continued

	Level	Least Sq Mean	Std Error	Level	Least Sq Mean	Std Error
evalOpts*retrain	0,0	39439622	597681.02	0,0	2.6638e+40	1.4524e+39
	0,1	35881417	594652.87	0,1	1.4611e+40	1.4524e+39
	0.1,0	39424120	594652.87	0.1,0	2.6638e+40	1.4524e+39
	0.1,1	35439762	594652.87	0.1,1	1.41e+40	1.4524e+39
	0.25,0	39424120	594652.87	0.25,0	2.6633e+40	1.4524e+39
	0.25,1	35725144	594652.87	0.25,1	1.6613e+40	1.4524e+39
	0.5,0	39424120	594652.87	0.5,0	2.6619e+40	1.4524e+39
	0.5,1	35033789	594652.87	0.5,1	1.5119e+40	1.4524e+39
	1,0	37453593	594652.87	1,0	1.3841e+40	1.4524e+39
	1,1	-1	594652.87	1,1	5.3525e+26	1.4524e+39

The amount of time that the true problem is utilized instead of the metal-model, or *evalOpts*, has an interesting effect. The optimization algorithm was able to find a better solution based on accessing the true problem less frequently. There is no conclusive evidence that there is an optimal ratio of evaluations that using the meta-model allows the algorithm to find a better solution in place of the true simulation. It can be speculated that this is due to the meta-model allowing the optimization algorithm to accelerate to the optimal region by leaving out the more complex attributes of the actual problem. Likewise, the interaction between the training population size and the amount of time the true problem is evaluated is important as well. Retraining the meta-model between iterations provides better solutions for the Schwefel problem but not for the Brown problem. This may be contributed to the complexity of the brown problem, again.

3.4.2 Regression

The regression data is similar to the neural network data in that it is difficult for the regression modeling technique to model Brown's almost linear function. This makes sense; least squares regression is in fact designed to model linear problems. This is exhibited in the quantiles presented in Table 3.5. This creates all sorts of problems regarding significance testing of parameters. For this reason, I have decided only to deal with the Schwefel function this point forward in the regression modeling technique. The significance values for the Schwefel function are summarized in Table 3.6: Regression Significance (significant values high-lighted in yellow). The significance levels for the Brown function are also included for your reference. Interactions were limited to 3rd order or less to limit the complexity of the analysis.

Table 3.5: Quantile Analysis of Regression Modeling Technique

Quantiles		Schwefel			Brown		
		2	10	20	2	10	20
100.00%	maximum	-1	-1.0036	-1.4917	-1	-0.78083	-15.7864
99.50%		-1	-1.024	-1.685	-1	-2.31522	-33.0203
97.50%		-1	-1.0458	-3.375	-1	-5.6675	-50.8247
90.00%		-1	-1.1874	-11	-1	-30.2158	-154.974
75.00%	quartile	-1	-2.2063	-58	-7.1	-5480.95	-9.36E32
50.00%	median	-1	-23	-251	-11	-169959	-3.86E53
25.00%	quartile	-1	-176	-805	-11	-1.89E14	-1.74E60
10.00%		-143.9	-756	-1698	-97	-1.31E28	-6.57E64
2.50%		-152.7	-780	-1874	-97.1	-1.13E32	-1.06E70
0.50%		-152.7	-868	-1951	-309.3	-9.98E35	-9.55E74
0.00%	minimum	-152.7	-1106	-2179	-310.1	-4.31E36	-1.87E78

Table 3.6: Regression Significance (significant values high-lighted in yellow)

			Schwefel				Brown		
			2	10	20		2	10	20
TermOpts	1	1	0.9891	0.0000	0.0000		0.9067	0.9742	0.0000
numTrain	1	1	0.8243	0.1152	0.1891		0.0001	0.0921	0.0000
p_val	1	1	0.6028	0.9478	0.7281		0.9096	0.0535	0.0000
evalOpts	1	1	0.0000	0.0000	0.0000		0.0000	0.0667	0.0000
retrain	1	1	0.0000	0.0000	0.0000		0.0000	0.0277	0.0000
TermOpts**numTrain	1	1	0.9883	0.6749	0.5690		0.9671	0.9992	0.0000
TermOpts*p_val	1	1	1.0000	0.9750	0.8601		1.0000	0.9932	0.0000
TermOpts**evalOpts	1	1	0.9918	0.0300	0.0567		0.8363	0.9552	0.0000
TermOpts*retrain	1	1	0.9915	0.6569	0.3200		0.9067	0.9747	0.0000
numTrain*p_val	1	1	0.5631	0.7650	0.1872		0.9226	0.1015	0.0000
numTrain**evalOpts	1	1	0.8151	0.1606	0.1488		0.0071	0.8459	0.0000
numTrain*retrain	1	1	0.9471	0.3330	0.1384		0.0073	0.0292	0.0000
p_val****evalOpts***	1	1	0.5900	0.9827	0.7300		0.9601	0.2121	0.0000
p_val*retrain	1	1	0.6030	0.8956	0.9295		0.9096	0.1374	0.0000
evalOpts*retrain	1	1	0.0000	0.0000	0.0000		0.0000	0.1414	0.0000
TermOpts**numTrain*p_val	1	1	0.9999	0.9317	0.8273		1.0000	0.9963	0.0000
TermOpts**numTrain**evalOpts	1	1	0.9912	0.9106	0.6759		0.9420	0.9987	0.0000
TermOpts**numTrain*retrain	1	1	0.9895	0.9540	0.8960		0.9671	0.9993	0.0000
TermOpts*p_val****evalOpts***	1	1	1.0000	0.9612	0.9294		1.0000	0.9875	0.0000
TermOpts*p_val*retrain	1	1	1.0000	0.9707	0.6135		1.0000	0.9928	0.0000
TermOpts**evalOpts*retrain	1	1	0.9936	0.8516	0.6419		0.8363	0.9552	0.0000
numTrain*p_val****evalOpts***	1	1	0.5493	0.7573	0.2680		0.9658	0.4667	0.0000
numTrain*p_val*retrain	1	1	0.5634	0.7048	0.7383		0.9226	0.1912	0.0000
numTrain**evalOpts*retrain	1	1	0.9475	0.2119	0.2164		0.0047	0.4467	0.0000
p_val****evalOpts****retrain	1	1	0.5902	0.9466	0.9812		0.9601	0.7974	0.0000

Table 4.8 indicates that both the amount of time that the true function is utilized and whether the meta-model is trained are significant parameters. As the complexity of the function increases then the amount of time the algorithm is allowed to work, ***TermOpts***, is also

significant which is due to the fact that the increased amount of iterations is going to access the real function more often which allows it to work longer, thus finding a better solution. In fact, the model fitted to the output data is so poor that the software package (SAS) being utilized will not allow for effect plots for anything other than the retraining option. However, it is determined that retraining the meta-model will provide a better solution by an Analysis of Variance (ANOVA) test conducted on the retraining parameter as shown in Figure 3.9 and Table 3.7 for the two and 20 variable problems, but does not have a positive effect in the ten variable problem.

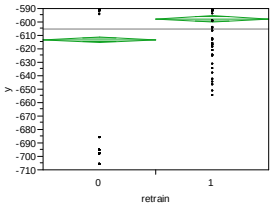
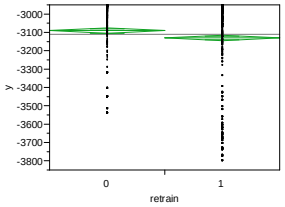
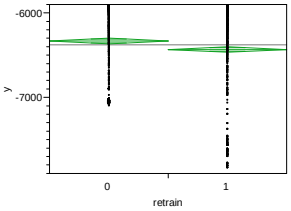
Schwefel – 2	Schwefel – 10	Schwefel – 20
		

Figure 3.9: ANOVA for Retrain

Table 3.7: ANOVA Values for Retrain

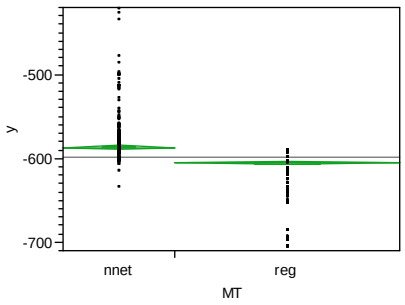
	Source	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
2	retrain	1	108415.7	108416	111.3212	<.0001
	Error	1798	1751072.7	974		
	C. Total	1799	1859488.5			
10	**Source**	**DF**	**Sum of Squares**	**Mean Square**	**F Ratio**	**Prob > F**
	retrain	1	688948	688948	12.7917	0.0004
	Error	1798	96838373	53859		
	C. Total	1799	97527321			
20	**Source**	**DF**	**Sum of Squares**	**Mean Square**	**F Ratio**	**Prob > F**
	retrain	1	4532109	4532109	16.8022	<.0001
	Error	1798	484978742	269732		
	C. Total	1799	489510852			

3.5 Neural-network Model versus the Regression Model

Figure 3.10 below shows that the neural-networking modeling method provides a better solution than the regression method, using the same experimental parameters for five out of six of the experiments. This is most likely due to the complexity and non-linearity of the problems that are being used as surrogates of the simulation model but also represents the complexity incurred in real simulation models.

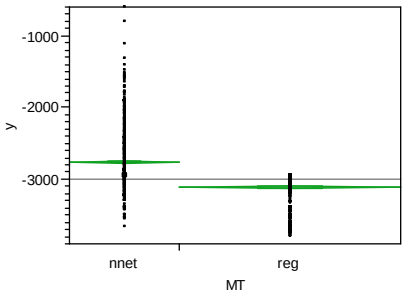
Figure 3.10: ANOVA for Neural-Network v Regression

Schwefel -2



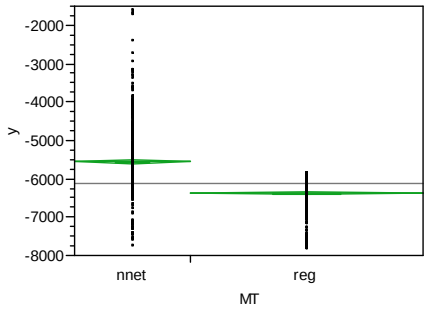
Source	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
MT	1	204639.7	204640	256.3916	<.0001
Error	2698	2153416.3	798		
C. Total	2699	2358056.0			

Schwefel – 10



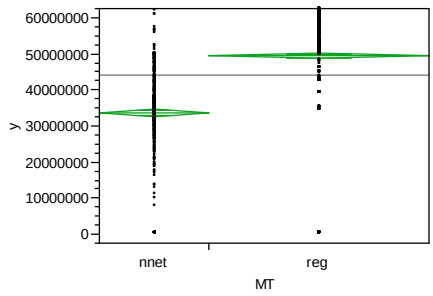
Source	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
MT	1	75448155	75448155	745.5761	<.0001
Error	2698	273022584	101194.43		
C. Total	2699	348470739			

Schwefel – 20



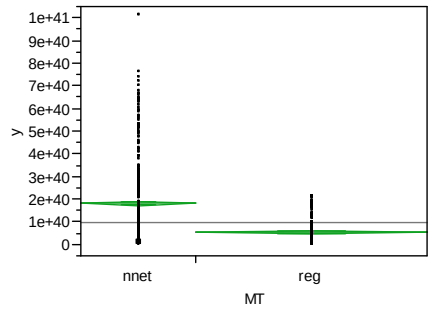
Source	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
MT	1	406132791	406132791	992.1459	<.0001
Error	2698	1104420495	409347.85		
C. Total	2699	1510553287			

Brown – 2



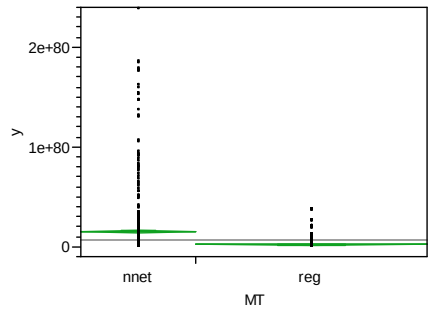
Source	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
MT	1	1.487e+17	1.487e+17	564.5264	<.0001
Error	2697	7.1041e+17	2.634e+14		
C. Total	2698	8.5911e+17			

Brown – 10



Source	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
MT	1	9.6707e+82	9.671e+82	635.8473	<.0001
Error	2698	4.1034e+83	1.521e+80		
C. Total	2699	5.0705e+83			

Brown – 20



Source	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
MT	1	1.015e+161	1.01e+161	384.2194	<.0001
Error	2698	7.125e+161	2.64e+158		
C. Total	2699	8.14e+161			

3.6 Discussion

The neural network modeling technique is much more effective for local modeling as seen in Figure 3.10. Utilizing the meta-modeling technique a portion of the time provides a better solution suggesting that the meta-model allows the searching algorithm to find the optimal region sooner (i.e., more efficiently) and creates individuals in that optimal region allowing for better exploration or that region. This explains why the amount of time that the algorithm is allowed to run has limited effect on the quality of the final solution. Retraining the data-set has an impact on the final solution but is not consistent from problem to problem. The previous analysis suggests that there is no optimal parameter setting that encompasses all problems, but rather the parameter settings that are best for that problem are specific to the problem being modeled. An experiment is needed to determine the effect of noise on the neural network modeling technique and the variables tested above. In the next step, stochastic noise will be added to the experiment for the neural-network portion of the experiment to evaluate the performance in an a more real-life simulation environment. It is worth noting that the regression modeling technique allowed the search algorithm to find the optimal region quickly. However, it did not possess the detail necessary to model complex problems.

4 Stochastic Experimental Procedure

From Chapter 3, the neural network modeling technique was shown to be more effective for local modeling at least for these sets of problems. The previous experiments utilized deterministic problems which are not equivalent to most stochastic simulations. Therefore the next step is to determine the effect noise has on the neural network modeling technique as seen in stochastic simulation models.

4.1 Goal

The second experiment will determine three objectives. The first objective is to verify the results found in Section 4. Additional information on the optimal solution was needed to be able to determine the optimal settings for the meta-modeling and search technique based on additional judgment criteria as mentioned in Chapter 3. These factors are the true solution values gathered from the true evaluation function, the number of simulations utilized when the optimal solution for that search replication is found, the number of times the meta-model is utilized for that same search replication, and the values for the predictive variables. Experimenting with these parameters will allow us, in Chapter 4 to verify the conclusions reached in the first set of experiments.

The second experimental object is to expand the number of objective functions from two to three. The experiment in this chapter will include the two evaluation functions from the first experiment (Schwefel's function and Brown's almost linear function) as well as the addition of the Corana function which is described in Figure 4.1 and Figure 4.2.

Corana Function

$$\theta(\mathbf{x}) = \begin{cases} 1 + \sum_{i=1}^{d} c_i(x_i - 1)^2, & \text{for } \mathbf{x} \in \Xi - \Upsilon, \\ 1 + \psi \sum_{i=1}^{d} c_i(z_i - 1)^2, & \text{for } \mathbf{x} \in \Upsilon, \end{cases}$$

where

$$z_i = \left\{ \begin{array}{ll} k_i s_i + t_i, & \text{if } k_i < 0, \\ 0, & \text{if } k_i = 0, \\ k_i s_i - t_i, & \text{if } k_i > 0, \end{array} \right\} \text{ for } i = 1, \ldots, d.$$

Figure 4.1: Corana Function Equation

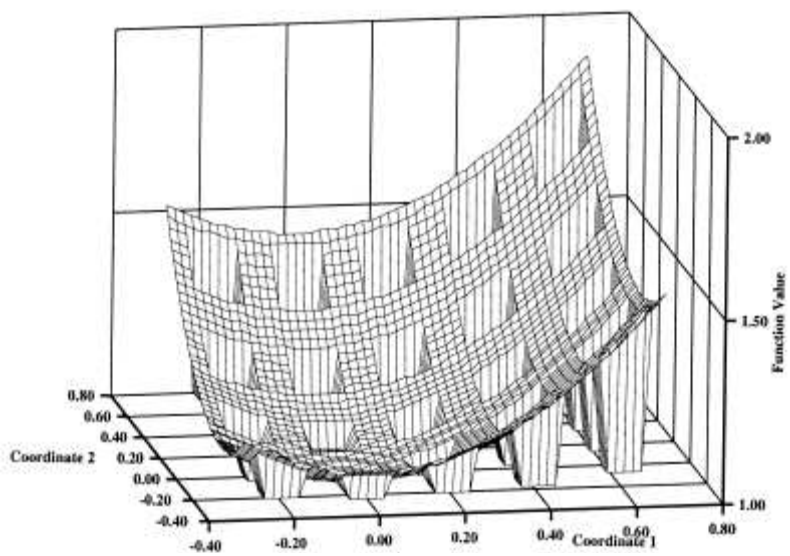


Figure 4.2: Corana Function Plot – n=2

The main experimental objective is to determine the effect that noise posses on the efficiency and effectiveness of the meta-modeling technique and the genetic algorithm searching technique. Adding noise has been found to be an adequate and effective method of impersonating the stochastic nature of a true simulation optimization problem [Humphrey and Wilson, 2000].

4.2 Experimental Setup

The experimental setup for the second experiment is quite similar to that of the first experiment which was done intentionally to allow for the direct comparison of the noise settings to those not containing noise and allow verification of the results concluded from the initial experiment. However, this experiment is limited to the neural network meta-modeling technique only since the stepwise regression meta-modeling technique was determined to be ineffective at least on this set of problems. Further discussion and explanation of this is done in the previous chapter.

To mimic the behavior of outputs generated by a stochastic simulation model, an additive white-noise error term is added to each of the outputs generated from the deterministic function where the error term is randomly sampled from a normal distribution with a mean of zero and standard deviation (σ) that is systematically varied to examine the effect of increasing levels of noise variability has on the optimization technique.

$$F_{\text{Stochastic}}(\boldsymbol{X}) = F_{\text{Deterministic}}(\boldsymbol{X}) + Normal(0, \sigma)$$

The noise component is added during the evaluation of the true evaluation function. In the true evaluation function, the function is evaluated fifteen times with the addition of a normally distributed random number with a mean of zero and a chosen standard deviation. The average of these fifteen evaluations is then taken as the value of the stochastic model. This is very similar to the method in which simulation optimization is conducted when a meta-modeling technique is not employed. The standard deviations are chosen based on a percentage of the optimal solution. The evaluation functions that are used in this experiment, Corana, Brown's and Schwefel's, are all programmed to have an optimal solution of negative one where the true optimal value of these functions is zero, however since the standard deviation is based on a percentage of the optimal value, zero would not have yielded any noise. The optimal value of negative one was chosen to coincide with that chosen by Humphrey and Wilson [2000].

4.3 Experimental Parameters

To test the GA's ability to be used as a simulation optimization technique and the effectiveness of utilizing a meta-model to increase the efficiency of the search owing to a decrease in the number of simulations that have to be run, a five parameter full factorial experiment will be conducted to determine the sensitivity of the number of decision variables, size of the data set needed to first build the meta-model, maximum number of generations to run the GA, and the number of times that the true evaluation is used versus the

meta-model, whether or not retrain the network and the amount of noise in the system. Many of the parameters are the same as the previous experiment: ***NumVar, NumTrain, TermOpts*** , ***Retrain*** and **TrueEval(*evalOpts*).** The new one variable is the ***Noise*** parameter.

Noise–Level of noise added to the true evaluation function. The addition of the noise parameter allows us to mimic a true stochastic simulation optimization problem. The noise associated with a simulation is best mimicked using a normally distributed random number with a mean of zero and a variance representative of the variance in the simulation. To mimic the procedure of a simulation optimization problem an average is taken which minimizes the variance from mean of the true value to that produced by the simulation. In this experiment, fifteen replications are performed on the true function with the addition of the noise and the average is then taken as the value that the genetic algorithm uses as its evaluation parameter. The parameter value passed to the GA is the value of the standard deviation. The standard deviation of zero is used to verify the results of the first experiment (i.e., represent the discrete experiment from Chapter 3). Table 4.1 shows the parameter levels and their values that will are used in the new experiment.

Table 4.1: Parameter Levels

Parameter	**Levels**				
NumVar	2	10	20		
NumTrain	200	500	1000		
TermOpts	100	300	600		
TrueEval	0	0.10	0.25	0.50	1.00
Retrain	0 – No	1 – Yes			
Noise Level	0	0.5	0.75	1	

4.4 Experimental Design

The experiment is a five factor full factorial design, meaning that all combinations of the parameters are distinct. This allows for a clear understanding of the effect that the parameters have on the final solution without being confounded with other variables. The five factor full factorial design is repeated on all three variable number settings and the three evaluation functions that are being studied. A tree representation of the experimental design assists to clarify the experimental structure, Figure 4.3.

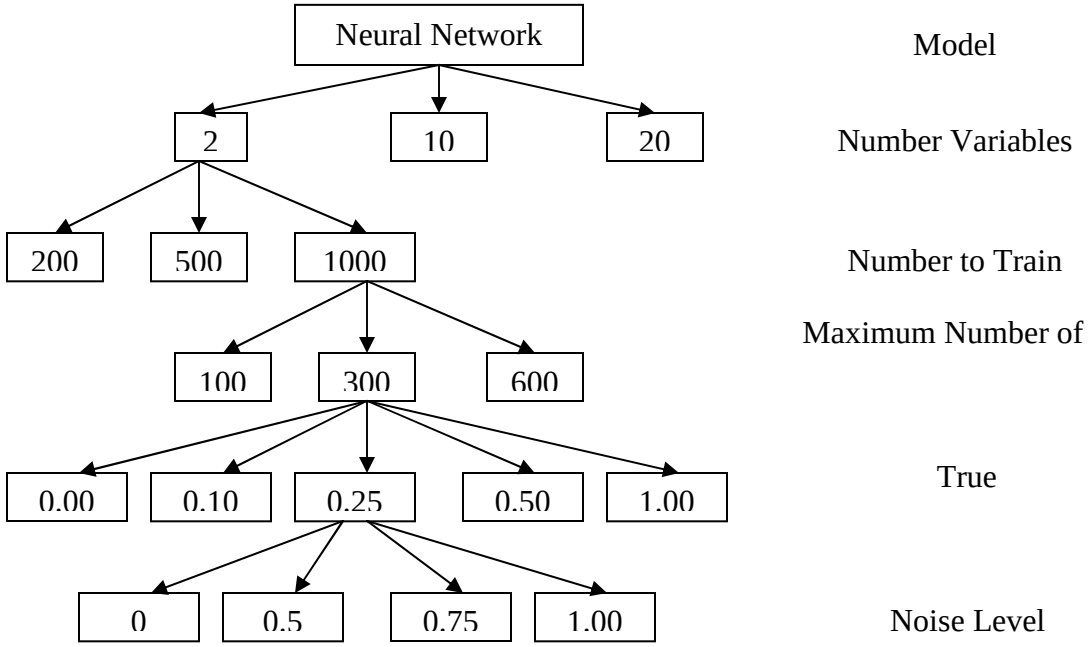


Figure 4.3: Experimental Design Structure

4.5 Output Data

Experiment two is designed and modified from lessons that were learned from experiment one. These lessons include that the experiment needs to store more data. Initially, in the first experiment, only the value for the best point was kept, on what generation of the genetic algorithm it was found, the number of true evaluations needed, and the number of meta-model evaluations that were done. These, being the number of meta-model and number of true evaluations are the total number of evaluations for the run. In this experiment the

number of meta-model and true evaluations on which the best value found is kept and stored in the data set. In addition, the values for the decision variables, the '*x*'s', are also kept. Using these actual decision variables the true value is found by evaluating the true evaluation function which allows for an easy calculation of the difference between the real function and the meta-model which allows a quick representation of the accuracy of the meta-model.

4.6 Results

The analysis will be split up by the three different functions (Schwefel, Brown, and Corana) which is extremely different from the results of Chapter 3. The primary output studied is the output value of the true function using the prediction variables found by the genetic algorithm during optimization search. The inclusion of noise in the experiment is designed to test the ability of the neural-network and genetic algorithm to ignore the noise thus providing the search algorithm a suitable alternative to the true value.

4.6.1 Schwefel Function

Table 4.2: Schwefel Significance Values

		Noise Levels			
2 Variables	**DF**	**0**	**0.5**	**0.75**	**1**
termOps	2	0.0841	0.0151	0.6615	0.7904
numTrain	2	0.0000	0.0000	0.0000	0.0000
termOps*****numTrain***	4	0.8522	0.1890	0.7372	0.5162
pctTrueEval	4	0.0000	0.0000	0.0000	0.0000
termOps*pctTrueEval	8	0.2537	0.6611	0.6819	0.6489
numTrain*pctTrueEval	8	0.0000	0.0000	0.0000	0.0000
termOps*****numTrain****pctTrueEval	16	0.8958	0.3518	0.1863	0.0130
retrain	1	0.0000	0.0000	0.0000	0.0000
termOps*retrain	2	0.0838	0.0148	0.6690	0.9198
numTrain*retrain	2	0.0000	0.0000	0.0000	0.0000
termOps*****numTrain****retrain	4	0.8516	0.1911	0.7382	0.6151

Table 4.2: Continued

pctTrueEval*retrain	4	0.0000	0.0000	0.0000	0.0000
termOps*pctTrueEval*retrain	8	0.2524	0.6595	0.6776	0.6891
numTrain*pctTrueEval*retrain	8	0.0000	0.0000	0.0000	0.0000
termOps****numTrain****pctTrueEval*retrain	16	0.8957	0.3479	0.1903	0.0066
10 Variables					
termOps	2	0.3181	0.1970	0.9848	0.3392
numTrain	2	0.0000	0.0000	0.0000	0.0000
termOps****numTrain***	4	0.9182	0.9528	0.8861	0.1302
pctTrueEval	4	0.0000	0.0000	0.0000	0.0000
termOps*pctTrueEval	8	0.3111	0.3416	0.7488	0.2769
numTrain*pctTrueEval	8	0.0000	0.0000	0.0000	0.0000
termOps****numTrain****pctTrueEval	16	0.3604	0.5983	0.9994	0.7096
retrain	1	0.0010	0.0000	0.0001	0.0004
termOps*retrain	2	0.3900	0.4584	0.8540	0.9059
numTrain*retrain	2	0.0000	0.0000	0.0000	0.0000
termOps****numTrain****retrain	4	0.8664	0.9638	0.9895	0.6811
pctTrueEval*retrain	4	0.0303	0.0011	0.0770	0.2518
termOps*pctTrueEval*retrain	8	0.2706	0.8389	0.8266	0.5542
numTrain*pctTrueEval*retrain	8	0.0000	0.0000	0.0000	0.0000
termOps****numTrain****pctTrueEval*retrain	16	0.2391	0.9896	0.7607	0.9719
20 Variables					
termOps	2	0.9004	0.2847	0.6964	0.7969
numTrain	2	0.0000	0.0000	0.0000	0.0000
termOps****numTrain***	4	0.8804	0.7262	0.9714	0.9030
pctTrueEval	4	0.0000	0.0000	0.0000	0.0000
termOps*pctTrueEval	8	0.8027	0.1415	0.7321	0.7669
numTrain*pctTrueEval	8	0.0002	0.0034	0.0000	0.0008
termOps****numTrain****pctTrueEval	16	0.9574	0.0611	0.9935	0.9995
retrain	1	0.0168	0.0019	0.0000	0.0175
termOps*retrain	2	0.6995	0.2951	0.8336	0.5963
numTrain*retrain	2	0.7729	0.4427	0.0118	0.6881
termOps****numTrain****retrain	4	0.8504	0.8224	0.6277	0.9443
pctTrueEval*retrain	4	0.5446	0.1454	0.0576	0.0356
termOps*pctTrueEval*retrain	8	0.9257	0.4054	0.9959	0.7916
numTrain*pctTrueEval*retrain	8	0.0036	0.3285	0.0001	0.0066
termOps****numTrain****pctTrueEval*retrain	16	0.9909	0.8361	0.9811	0.9997

Table 4.2 shows that there is not a significant difference in the importance of the parameters with an increase in noise levels for the Schwefel function. An ANOVA on the noise levels against the best solution backs up this conclusion for all three functions: e Schwefel , Brown and Corana. As seen in Figure 4.4: ANOVA Noicse Level; none of the noise levels exhibit a

p-value that is below the 0.05 threshold necessary to be considered significant except for the 10 variable Corana setting. The *student –t* test shows a difference between the noise level in Table 4.4. The non-linear nature of the significant differences of the means suggests that this is an anomaly and not attributed to the increasing effect of noise on the final solution. The parameters highlighted in Table 4.2 are the same as those in Table 4.4 thereby allowing us to draw the same conclusions and thus analyze the effectiveness of the modeling through three vantage points: 1) accuracy of the model and 2) speed in which the best solution was found and 3) the amount of times the real function is utilized. This experiment was only designed to track the first two.

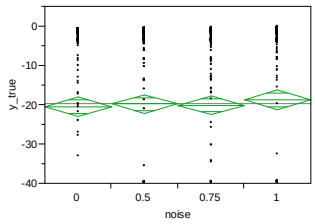
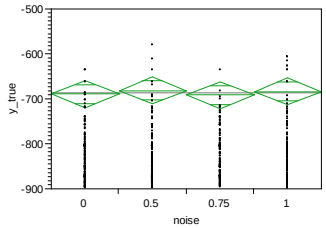
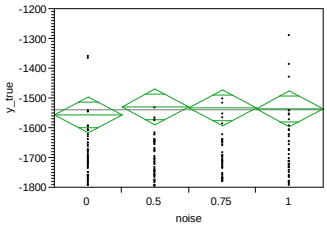
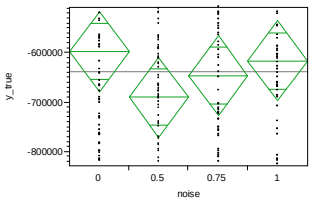
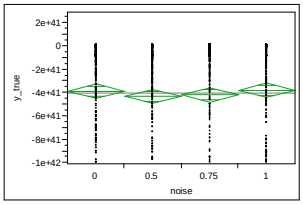
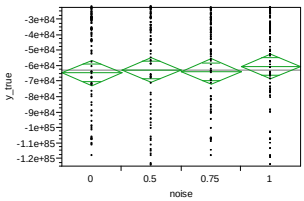
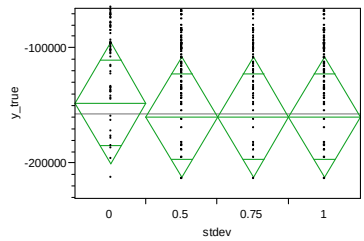
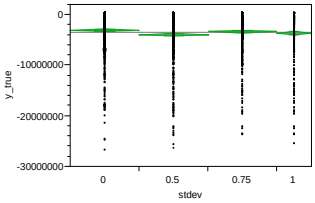
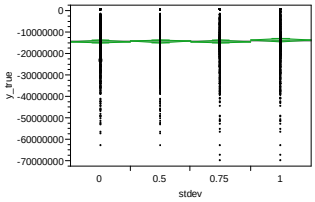
2	10	20
Schwefel Function		
		
P>F = 0.7625	P>F = 0.9663	P>F = 0.9338
Brown Function		
		
P>F = 0.4106	P>F = 0.5710	P>F = 0.9203
Corana Function		
		
P>F = 0.9841	P>F = 0.0002	P>F = 0.4048

Figure 4.4: ANOVA Noicse Level

Table 4.3: Corana 10 variable Student T means analysis

Level	- Level	Difference	Lower CL	Upper CL	p-Value
0	0.5	892235.9	479266	1305206	<.0001
0.75	0.5	610335.0	196439	1024231	0.0039
0	1	582594.3	77139	1088049	0.0239
1	0.5	309641.6	-195907	815191	0.2299
0.75	1	300693.4	-205518	806905	0.2442
0	0.75	281900.9	-131880	695682	0.1817

Table 4.4: Mean Comparison

Level				Mean
0	A			-3104779
0.75	A	B		-3386680
1		B	C	-3687373
0.5			C	-3997015

Levels not connected by same letter are significantly different.

4.6.2 Model Accuracy

The performance of a meta-modeling technique is partially based on its ability to model the true problem accurately, or well enough to find the solution region. A correlation analysis between the actual value and the estimated value is summarized in Table 4.5 below in terms of the R-squared values. If the neural-network were perfect and predicted the exact value every time then the R-squared value would be one. The low values in Table 4.5 show that the correlation is weak and thus that the neural network is doing a poor job of modeling the true function. This is reinforced by a distribution analysis in Table 4.6 showing the

difference between the predicted and actual values are acceptable for the two variable function, but get exponentially greater as the more complexity is introduced.

Table 4.5: R-squared Values of Estimate & Actual Correlation

2	10	20
Schwefel		
0.2441	0.3740	0.1309
Brown		
0.0246	0.0029	0.0060
Corana		
0.2076	0.3266	0.2076

Table 4.6: Schwefel Est v Actual differnece Quantitiel analysis

		Schwefel		
		2	10	20
100.00%	maximum	19.20	0.34	0.50
99.50%		8.89	0.07	0.24
97.50%		0.00	0.00	0.00
90.00%		0.00	-0.21	-0.07
75.00%	quartile	-0.41	-0.75	-0.71
50.00%	median	-0.98	-1092.86	-2196.13
25.00%	quartile	-12.29	-1666.67	-2709.29
10.00%		-89.39	-1798.17	-3173.10
2.50%		-178.27	-2182.72	-4204.08
0.50%		-228.94	-2858.91	-5143.48
0.00%	minimum	-315.64	-3372.99	-7605.51
Mean		-24.19	-1000.90	-1843.26
Std Dev		48.08	759.17	1304.91
Std Err Mean		0.80	12.65	21.75
upper 95% Mean		-22.62	-976.09	-1800.62
lower 95% Mean		-25.76	-1025.70	-1885.90
N		3600.00	3600.00	3600.00

It is not necessary for the neural network to be able to model the function completely, but rather provide a representative model of the overall structure of the problem. This will allow the genetic algorithm to find the optimal region where the best solutions lie and begin searching within that area. The ability of the function to find the optimal values as shown in quantile analysis in Table 4.6 demonstrates that the function is able to find the optimal solution or very close to it 25% of the time. It is interesting that this is the case since the experimental design has the true function utilized exclusively for 20% of the data. One could draw the conclusion that the good solutions are due to the experimental settings that contains exclusive use of the true function. The subsequent analysis this is shown not to be the case.

Table 4.7: Best Solution Quantile Analysis - Schwefel

		Schwefel		
		2	10	20
100.00%	maximum	-0.30	-0.84	-1.49
99.50%		-0.65	-1.03	-2.97
97.50%		-0.79	-1.16	-5.41
90.00%		-0.97	-1.92	-15.65
75.00%	quartile	-1.01	-9.60	-236.92
50.00%	median	-1.39	-916.05	-1990.93
25.00%	quartile	-4.16	-1042.51	-2194.56
10.00%		-77.56	-1101.98	-2339.67
2.50%		-152.52	-1277.43	-2552.57
0.50%		-152.84	-1336.47	-2696.49
0.00%	minimum	-153.08	-1488.64	-2793.39
Mean		-19.82	-686.69	-1539.55
Std Dev		36.55	460.85	926.60
Std Err Mean		0.61	7.68	15.44
upper 95% Mean		-18.62	-671.63	-1509.28
lower 95% Mean		-21.01	-701.75	-1569.83
N		3600	3600	3600

Table 4.8: Best Solution Quantile Analysis – Brown

		Brown's		
		2	**10**	**20**
100.00%	maximum	1.519189	-0.780828	-12.71154
99.50%		0.363626	-2.137987	-22.14636
97.50%		-0.57081	-4.961792	-48.76714
90.00%		-1.00045	-28.09074	-154.3678
75.00%	quartile	-335.049	-8.35E+38	-1.07E+81
50.00%	median	-139500	-1.52E+40	-2.5E+83
25.00%	quartile	-808682	-1.28E+41	-3.63E+84
10.00%		-2033137	-1.55E+42	-2.57E+85
2.50%		-3970010	-3.69E+42	-4.74E+85
0.50%		-7939847	-3.69E+42	-5.82E+85
0.00%	minimum	-1.8E+07	-3.69E+42	-5.82E+85
Mean		-637923	-4.06E+41	-6.31E+84
Std Dev		1232633	9.309E+41	1.247E+85
Std Err Mean		20546.73	1.551E+40	2.079E+83
upper 95% Mean		-597639	-3.75E+41	-5.9E+84
lower 95% Mean		-678208	-4.36E+41	-6.72E+84
N		3599	3600	3600

Table 4.9: Best Solution Quantile Analysis - Corana

		Corana		
		2	**10**	**20**
100.00%	maximum	0	0	-0.00046
99.50%		0	0	-1.00825
97.50%		0	-0.16275	-411.631
90.00%		0	-30.426	-4487.21
75.00%	quartile	-4257.18	-18907.2	-4541891
50.00%	median	-10447.7	-1867910	-1.4E+07
25.00%	quartile	-47687.2	-5114668	-2E+07
10.00%		-104540	-8331208	-2.9E+07
2.50%		-3379603	-1.8E+07	-4.1E+07
0.50%		-3380628	-2.4E+07	-5.4E+07
0.00%	minimum	-2E+07	-3.9E+07	-7E+07
Mean		-157076	-3523698	-1.4E+07
Std Dev		802169.4	4475926	11496659
Std Err Mean		13375.07	79901.77	191957.9
upper 95% Mean		-130853	-3367033	-1.4E+07
lower 95% Mean		-183300	-3680363	-1.5E+07
N		3597	3138	3587

Thereby we are able to conclude that the neural-network does not do a good job of modeling the data over a small solution space, but does allow the search algorithm to find the solution area effectively. This ability is deteriorated as the complexity of the function increases as shown in Table 4.7, Table 4.8, and Table 4.9. The drastic increase in problem complexity was not mirrored by an increase in the modeling or search parameters; such as population size, neural network complexity, or genetic algorithm generations.

.

4.6.3 Number of Generations for Best Solution

How quickly the search algorithm is able to find the best solution is critical in limiting the number of times the true simulation is used; thus reducing the computation cost of the optimization. The experimental set-up dictates the total number of times the simulation is accessed based on the parameter settings and the generation in which the best solution will be analyzed. An ANOVA is conducted on the individual parameters with respect to the generation number that results in the best solution as seen in Figure 4.5, Figure 4.6, and Figure 4.7. It is obvious that the longer the search algorithm is allowed to run, the better the solution will be. The current experiment was not set-up to identify how much better the solution was only how quickly it was found. Although the size of the training population does have an effect; the effect is not consistent between the different function complexities. An increase in the training size may not be within the 0.05% probability of being caused by noise to be considered statistically significant; however, the mean generation number for the best solution is consistently lower with the higher population size. The same holds true for the amount of time the true function is evaluated. It is worth noting that the mean of the 50% true evaluation average is higher than the others but there is no statistical difference between the 25% and 100% true function evaluations. This suggests that using the true evaluation slightly will allow the search algorithm to quickly find the optimal solution space which may be due to the simplification of the true evaluation function by the meta-model. The results from the previous chapter show that using the true evaluation function provides a better

solution. A progressive increase in the amount of time that the true evaluation function is increased could take advantage of this fact. The retraining parameter does help determine how quickly the best solution is found. Retraining the data allows the algorithm to find the solution quicker for both the Schwefel and Brown problems but not for the Corana problem which is quite difficult.

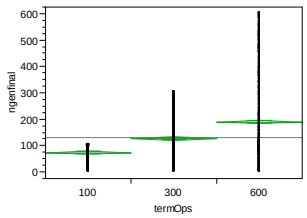
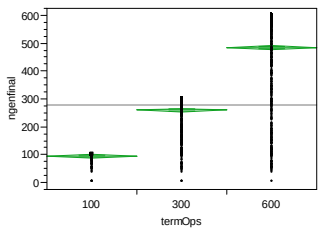
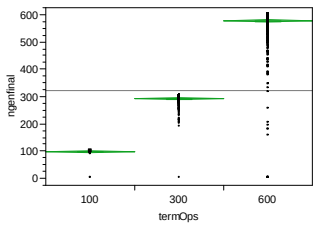
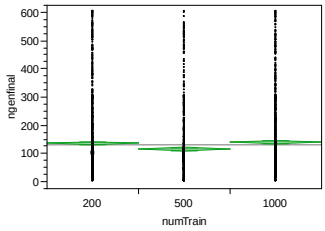
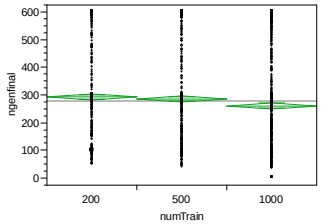
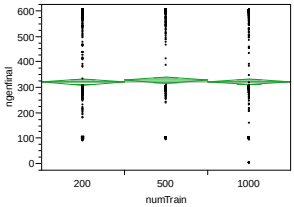
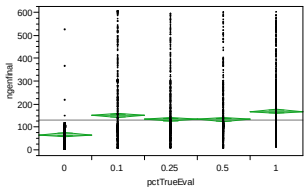
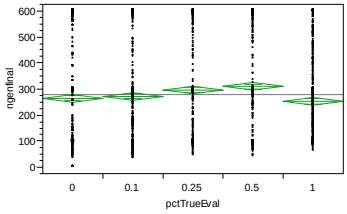
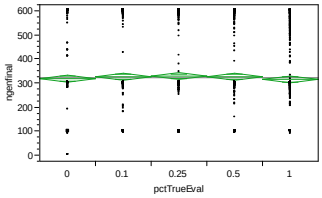
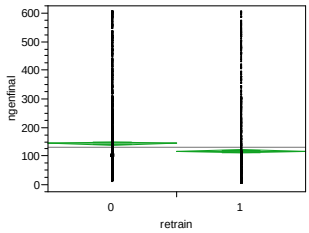
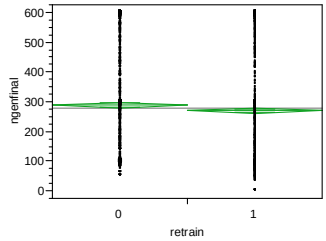
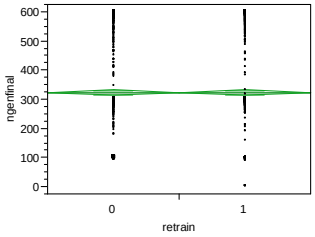
	2	10	20
TermOpts			
	Prob >F <0.0001	Prob >F <0.0001	Prob >F = 0.000
numTrain			
	Prob >F <0.001	Prob >F – 0.0003	Prob >F = 0.5939
evalOpts			
	Prob >F <0.001	Prob >F <0.0001	Prob >F = 0.7578
retrain			
	Prob >F <0.001	Prob >F = 0.0038	Prob >F = 0.9882

Figure 4.5: ANOVA for Parameters – Schwefel

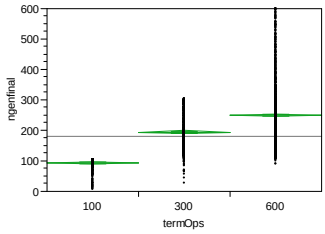
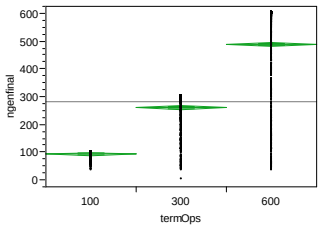
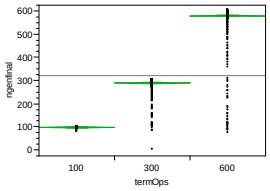
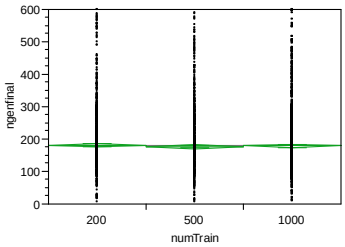
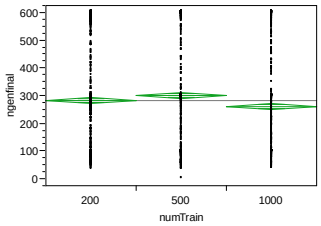
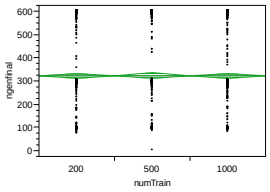
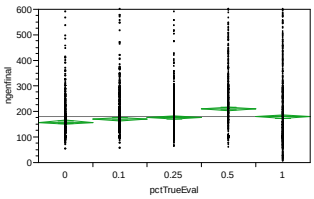
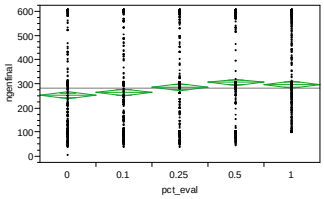
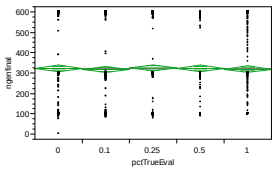
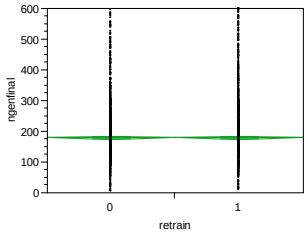
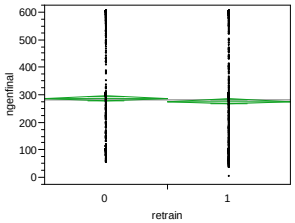
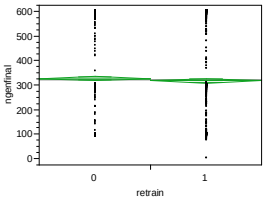
	2	10	20
TermOpts			
	Prob >F <0.0000	Prob >F <0.0001	Prob >F = 0.000
numTrain			
	Prob >F <0.5536	Prob >F < 0.0001	Prob >F = 0.9847
evalOpts			
	Prob >F <0.001	Prob >F <0.0001	Prob >F = 0.9738
retrain			
	Prob >F = 0.8701	Prob >F = 0.0849	Prob >F = 0.1734

Figure 4.6: ANOVA for Parameters – Brown

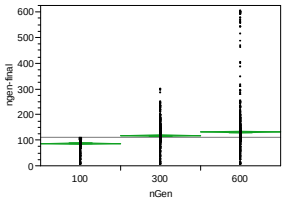
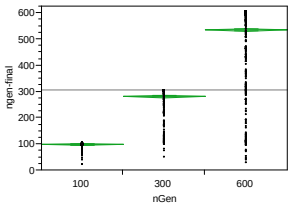
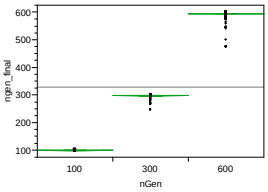
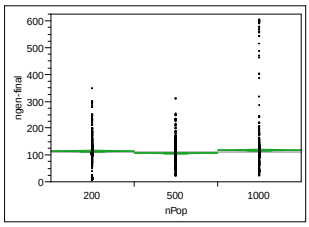
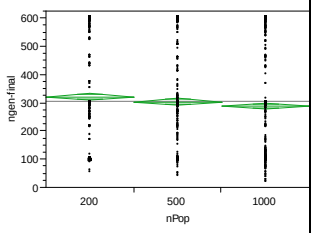
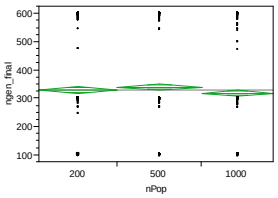
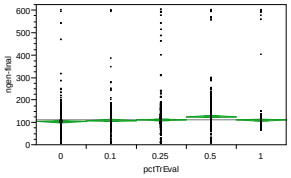
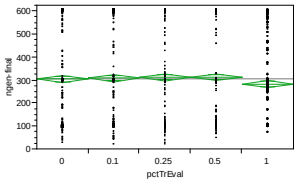
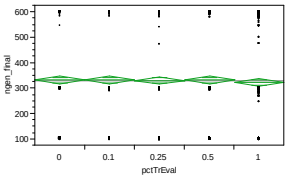
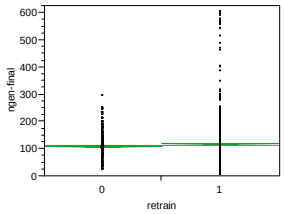
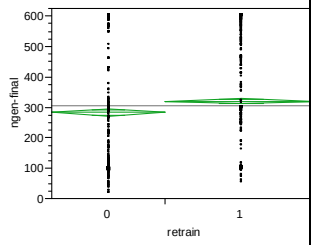
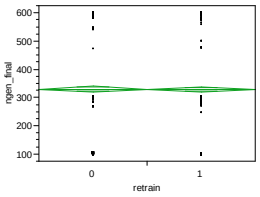
	2	10	20
TermOpts			
	Prob >F <0.0001	Prob >F <0.0001	Prob >F = 0.000
numTrain			
	Prob >F < 0.0001	Prob >F = 0.0007	Prob >F = 0.0435
evalOpts			
	Prob >F <0.001	Prob >F = 0.0449	Prob >F = 0.8940
retrain			
	Prob >F = 0.8701	Prob >F < 0.0001	Prob >F = 0.8081

Figure 4.7: ANOVA for Parameters - Corana

4.7 Experiment to test solution quality v the iteration of the solution

A simple experiment was conducted to determine of a combination of the meta-model and the true evaluation function generates a better solution in less true function evaluations. The Schwefel and Corana functions were chosen based on the ability of the meta-model to approximate the function.

Table 4.10 Speed to Optimal Solution

	Best Solution Found		**Number of Simulations Used When Best is Found**	
	2	**10**	**2**	**10**
Level	Mean	Mean	Mean	Mean
	Schwefel			
0.25	-63.111	-991	15492	116072
0.5	-44.762	-916.7	13071	96947
0.75	-79.71	-984.1	8496	86252
1	-55.298	-1024.8	13470	85854
	Corana			
0.25	-1246810	-3335034	28471	1151273
0.5	-425829	-3224872	59712	1723694
0.75	-955325	-6778524	46653	1206110
1	-1221155	-9310869	26824	1380108

Table 4.10 shows that using the meta-model a portion of the time allows for us to find a solution as good if not better in a substantially shorter period of time.

4.8 Conclusions

Noise poses no effect on the ability of the function to find the best solution so the conclusions reached in Chapter 3 hold true as well. The additional information collected in this experiment allows the determination of effectiveness of the modeling technique and how quickly the search algorithm is able to find the best solution. As pointed out in Section 4.6.2, the neural-network does not provide an accurate modeling technique but is able to fit well enough for the search algorithm to find the optimal solution area.

Retraining the model allows the search algorithm to find the optimal solution quicker. Although the other parameters appear to be significant, the optimal setting across the different function complexities is not consistent. It was also found that if the search algorithm is allowed to search longer, a better solution will most likely be found. An adjustment to future experiments will allow us to track the best solution of each generation.

The complexity of the model and effective time of the search technique do not increase in the same manner as the complexity of the functions being evaluated. Therefore, the parameter settings are not adequate to determine the optimal solution. The effects of the parameters tested are analyzed in this small range and are effective for the two and ten variable problems. Their effects can be extrapolated out to the functions with the greater complexity.

5 Conclusions and Future Work

The research shows that although the neural network does not do a good job of modeling the tested function exactly, it does model it with the generality that allows the search algorithm to find the region that holds the optimal or near optimal solution. Chapter 3 shows that the parameters tested do not have an effect on the final solution but in Chapter 4 it was shown that these factors do contribute to the speed in which that solution is found owing to the noise. Retraining the meta-model periodically adds to the effectiveness since new points are added as the search progresses. The effectiveness of the search algorithm can be enhanced by increasing the complexity of the meta-model, the search population and the amount of time the algorithm is allowed to operate in-line with the increase in complexity of evaluation functions.

The linear regression meta-modeling technique provides a broad picture of the solution landscape. The neural-network provided a better local area modeling technique with the true function providing the best for pin-point evaluation. A progressive combination of the three could provide the benefits of all of them. It may inhibit the global search ability of the algorithm. In this case, it would be necessary to reduce the possible solution space as the search algorithm hones in on the optimal solution area. This coupled with a progressive increase in the amount of time the true evaluation function is utilized can reduce the computational cost of the optimization.

Future work should include a combination of meta-modeling techniques to leverage the strengths of each while minimizing their weaknesses. Further research needs to be conducted to identify when these meta-model techniques should be adjusted to a progressively more intense local modeling technique. It has been identified that the longer the algorithm is allowed to run the better the solution it gets. Further research is needed to determine how quickly the meta-modeling technique reaches a good solution. This can act as the identifier to being the new modeling method. It was also shown that utilizing the true value in conjunction with the meta-model provides a solution quicker and with true evaluations needed. There is an inevitable computational expense associated with simulation optimization. Can the optimization algorithm optimize the use of this expense by loading it into the end of the search? A progressive increase in the percentage of time that the true evaluation is utilized needs to be studied where an exponential increase as opposed to a proportionate one could potentially allow for the optimization algorithm to utilize the strengths of the meta-modeling techniques.

REFERENCES

Andradottir, Sigrun. "A Review of Simulation Optimization Techniques." Proceedings of the 1998 Winter Simulation Conference. 1998: 151-158.

Avello, Eduardo A., Felipe F. Baesler, Reinaldo J. Moraga. "A Meta-Heuristic based on Simulated Annealing for Solving Multi-Objective Problems in Simulation Optimization." Proceedings of the 2004 Winter Simulation Conference. 2004: 508-513.

Carson II, John S. "Introduction to modeling and simulation" Proceedings of the 2003 Winter Simulation Conference. 2003: 7-13.

Humphrey, David G., James R. Wilson. "A Revised Simplex Search Procedure for Stochastic Simulation Response Surface Optimization." INFORMS Journal on Computing, Vol. 12, No. 4, Fall 2000: 272-283.

Ifram: http://irafm.osu.cz/en/c43_developing-new-evolutionary-algorithms-for-global-optimization-in-fuzzy-models/

Joshi, Shirish, Hanif D. Sherali, and Jeffrey D. Tew. "An Enhanced Response Surface Methodology (RSM) Algorithm Using Gradient Deflection and Second-Order Search Strategies." Computers Ops Res. Vol 25, No 7/8, 1998: 531-541.

Li, Shuhui. "Comparative Analysis of Backpropagation and Extended Kalman Filter in Pattern and Batch Forms for Training Neural Networks". Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on
Volume 1, 15-19 July 2001 Page(s):144 - 149 vol.1

Magoulas, Geourge D., tillal Eldabi, Ray J. Paul "Global Search Strategies for Simulation Optimisation". Proceedings of the 2002 Winter Simulation Conference. 2002: 1978-1985.

Matlab 2009.
http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/index.html?/access/helpdesk/help/toolbox/nnet/backpro7.html&http://www.google.com/search?hl=en&rlz=1T4GGLL_enUS306US306&q=scaled+conjugate+gradient+algorithm+backpropagation&aq=f&oq=&aqi=

Michalewicz, Zbigniew. Genetic Algorithms + Data Structures = Evolutionary Programs Springer-Verlag Berlin Heidelberg New York, 1996.

Neddermeijer, H. Gonda, Gerrit J van Oortmarssen, Nanda Piersma, Rommert Dekker. ”A Framework for Response Surface Methodology for Simulation Optimization”. Proceedings of the 2000 Winter Simulation Conference. 2000: 129-136

Olafsson, Sigurdur, Jumi Kim “Simulation Optimization” Proceedings of the 2002 Winter Simulation Conference. 2002: 79-84.

Rojas, Raul. *Neural Networks*. Springer-Verlag, Berlin, 1996

Santos, M. Isabel Reis dos, Acacio M. O. Porta Nova. “The main issues in non-linear simulation metamodel estimation.” Proceedings of the 1999 Winter Simulation Conference. 1999: 502-509

Stuckman, B.E., B. Nelson, W.d. Kelton, G. M. Clark. “Comparison of global search methods for design optimization using simulation.” Proceedings of the 1991 Winter Simulation Conference. 1991: 937-944

Wackerly, Dennis D., William Mendenhall III, Richard L. Scheaffer. Mathematical Statistics with Applications. Duxbury, 2002.