Time and Accuracy tradeoff using Artificial Neural Nets and Genetic Algorithms

by

Christopher Tan

B.S., Mechanical Engineering
University of Illinois at Urbana-Champaign, 1998

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

at the

Massachusetts Institute of Technology

Feb 2000

Signature of Author ....
Department of Mechanical Engineering
Feb 1st, 2000

Certified by: ....
David Wallace
Ester and Harold Edgerton Associate Professor of Mechanical Engineering
Thesis Supervisor

Accepted by: ....
Ain A. Sonin
Chairman, Department Committee on Graduate Students

Time and Accuracy tradeoff using Artificial Neural Nets and Genetic Algorithms

by

Christopher Tan

Submitted to the Department of Mechanical Engineering
On Feb 1st, 2000 in Partial Fulfillment of the
Requirements for the Degree of Master of Science

ABSTRACT
The effectiveness of using Artificial Neural Networks (ANNs) to substitute for slow function evaluations within a Genetic Algorithm (GA) optimization is investigated. A framework that relates the accuracy of the neural nets to their extent of use during optimization was developed and tested. Population statistics are also investigated as possible indicators in determining the extent to which neural networks might be reliably used. Potential methods of variable population size and time-related enhancements that optimize the usage of neural nets during optimization are discussed.

Results show that single neural nets perform effectively during optimization, producing scores very close to those obtained with a normal GA as long as the accuracy of the nets are within certain bounds. Less accurate ANNs still within certain criteria gave accurate and faster optimizations when used in combination with the real models during optimization. This result worked well for single and multiple ANNs for different sub-models. There was no absolute ANN accuracy value that could be used to guarantee good results, and the optimization accuracy was dependent on the complexity of the function. Population statistics were found not to be instructive in determining when to stop using the nets, but increasing population size during ANN use produced better results at a low computational cost.

Thesis Supervisor: David Wallace
Title: Ester and Harold Edgerton Associate Professor of Mechanical Engineering

Acknowledgements

I would like to thank my sponsor company, the Singapore Economic Development Board, for their financial aid that supported this thesis. Also, many thanks to David Wallace, my thesis advisor, for the numerous pointers that guided my research, without which I would have spent additional time pursuing fruitless approaches, and most of all, for being a listening ear and hearing out my proposals. Many thanks also to the people whose base code has made this possible : Matthew Wall's excellent GALib and Nicola Senin's DOME implementation of it and the time he took to respond to my questions about the intricacies of his code in spite of being halfway across the world in Italy. My thanks goes to Juan Deniz, not just for writing the robust Neural Net code, but also as a source of help and a friend throughout the progress of this thesis. My appreciation also to the wonderful family at the MIT CADLAB, especially the talented and helpful Nick Borland, for great memories and help in coding and solving everyday problems. Finally, a big thank you to my parents back in Singapore who have always encouraged me to think independently and provided the room to broaden my horizons, without which this would not have been possible.

Table of Contents

Table of Figures

1. Introduction

Real -world design problems have become increasingly large and complex — computationally-demanding modeling techniques and simulation capabilities are being increasingly applied to sub-problems. The need to find overall optimal solutions for such complex problems taking into account trade-offs and constraints is important. Genetic Algorithms described in Chapter 2 have been found to perform such optimizations robustly (Senin et. al. 1999). However, as each of the sub-problems become more computationally demanding to solve, Genetic Algorithms run into problems as they usually require many objective evaluations. While being able to more accurately produce a truly optimal solution by considering every single factor input, optimization algorithms fall mercy to the speed at which each input is evaluated. Just a single slow sub-problem evaluation can bring the entire optimization routine down to a crawl.

Meta-models like Artificial Neural Networks (ANNs), described in Chapter 2, are able to simulate the behavior of such sub-problems by computing an output almost instantaneously when given the input parameters of each input with a larger model. The idea of using of neural networks in Genetic Algorithms (GAs) is promising in being able to trade-off large time savings in exchange for a degree of accuracy.

This thesis investigates the use of artificial neural networks to substitute for computationally intensive sub-problems. The neural nets are used to replace slow function evaluations and their outputs used in the objective function of a GA optimization.

But how does one determine the effectiveness of neural net use in GAs? How do we know how accurate a neural net has to be before we can use it for optimization? We first develop a framework in Chapter 3 to analyze the effectiveness of ANN use in GA optimization. Accuracy boundaries for a ANN that give good and poor performance in GAs are analyzed and a measure is found to compute the length of ANN use for optimal performance. Parameters like statistics of the GA population, service time and sensitivity are also looked into, though experiments are not done on all of these parameters. This will affect the decision to change over from using a neural net to the actual computation. Several suggestions like time dominance and time creep in using ANNs in GAs are also elaborated upon given instances of parallel computation of function evaluations.

Results in Chapter 4 show that ANNs give promising results very close to those produced by a normal optimization if the ANNs are accurate, but produce significantly different results with inaccurate neural networks. Inaccurate use produces results after many generations that are poorer than those produced in the first few generations of a normal optimization. There was no absolute ANN accuracy value (or mean-squared error) that could be used to guarantee the success of hybrid optimization. The framework that was developed to analyze the effectiveness of neural net use was shown to be very effective in deciding the extent of ANN usage given its accuracy during an optimization. This worked not only for single ANN use for one sub-problem, but also for multiple ANNs within an optimization as well. Population statistics were not shown to be effective in guiding neural net use or indicating a point during the optimization to turn off the neural net in favor of the true sub-problem model. Doubling the population during the ANN optimization phase also proved to be pleasantly effective in producing better results with low computational costs. The results of this thesis and some limitations on its use are discussed in Chapter 5. In addition there are recommendations for future work.

2. Background

2.1 Optimization and meta-modeling

In complex design models with multiple variables and function evaluations, there often exists a need to find an optimal design configuration given an objective function. Search and optimization techniques that are commonly used include calculus-based methods (hill-climbing), enumerative schemes (dynamic programming) and random search algorithms (simulated annealing) and evolutionary algorithms (genetic algorithms) (Goldberg 1989). **Genetic Algorithms** (GAs), which will be described in section 2.1, have been shown to be a robust optimization technique capable of both exploring and exploiting the search space for a given design problem by using the concept of evolution in searching for better solutions.

Optimization techniques usually require multiple function evaluations over a period of time before converging on an optimal solution, and the presence of functions that require high computational time to produce an output can significantly retard optimization speed. Product design problems involving CAD models, finite element analysis (FEA) and computational fluid dynamics (CFD) fall into this category (El-Beltagy, 1998 pp. 1; Deniz, 1998).

Meta-models are able to address this problem by simulating a function's output when given the input parameters (Simpson et al., 1997), either for the entire objective function, or for just the properties of an individual sub-problem. The aim of a meta-model is to construct an approximate representation of the model that requires only a small fraction of time to compute. Meta-modeling techniques that have been used include neural networks (Schalkoff 1997), response surface methodologies (Lorenzen T., 1993) kriging techniques (Ratle, 1998) and gaussian regression models (El-Beltagy, 1999).

Artificial neural networks (ANNs) techniques, described in section 2.2, are popular due to their lack of need for a pre-determined model (unlike regression). They have been used in signal processing, pattern recognition (handwriting, voice) and financial systems (Schalkoff 1997 pp. 6). GAs have also been commonly used to train ANNs, but the use of ANNs as meta-models in GAs is less common. One example of such use has been in groundwater management models (non-deterministic) for the location and pumping schedules of wells, with the resulting neural net model used in a genetic algorithm to find the optimal well configuration (Rogers 1992).

2.1 Genetic Algorithms

Genetic algorithms (GAs) have been in used successfully since the early 1970s to optimize different sets of problems. Invented by John Holland in the late 1960s at the University of Michigan, (Holland 1975), GAs use the principles of natural selection for their operational effectiveness. GAs offer much improved performance over more traditional optimization techniques such as linear programming, dynamic programming and simulated annealing (Beasley 1993) as they offer much more powerful exploration capabilities of the search space in addition to exploitation capabilities. As computation capabilities have increased, they have found applications in many practical design problems.

GAs have been shown to be effective in solving design problems (Senin et. al. 1999a,b) in the context of the DOME (Distributed Object Modeling Environment) project at the MIT CADLAB. This project has developed a crowding variation called the Struggle GA, which will be used for the tests in this thesis.

2.1.1 Review of how GAs work

Fig. 2.1 shows, in schematic form, how a simple GA works. The steps of encoding, fitness calculation, parent selection and reproduction are briefly reviewed here.

2.1.1.1 Encoding

The first step in creating a genetic algorithm for optimization purposes is to **encode** the problem. The design problem is broken down into a representation consisting of search variables (genes) that can be combined to form a genome. This may be done through a variety of methods. For real and integer number variables, **real** (direct) or **binary** encodings are most common. Direct encoding uses the actual value of the number as a parameter, whereas binary encoding uses strings of binary values to represent the value. Direct encoding has been shown to produce better results (Senin 1999) and we use such an encoding for our tests.

2.1.1.2 Fitness calculation

A measurement has to be made of how effective different combinations of the genome are when different values for the search variables are set. The fitness function computes this measurement of optimality. The fitness function relates the way the search variables interact to produce the fitness value of that particular individual genome. For maximization problems, a higher fitness is indicative of a better individual and for minimization, the converse of a lower fitness is true.

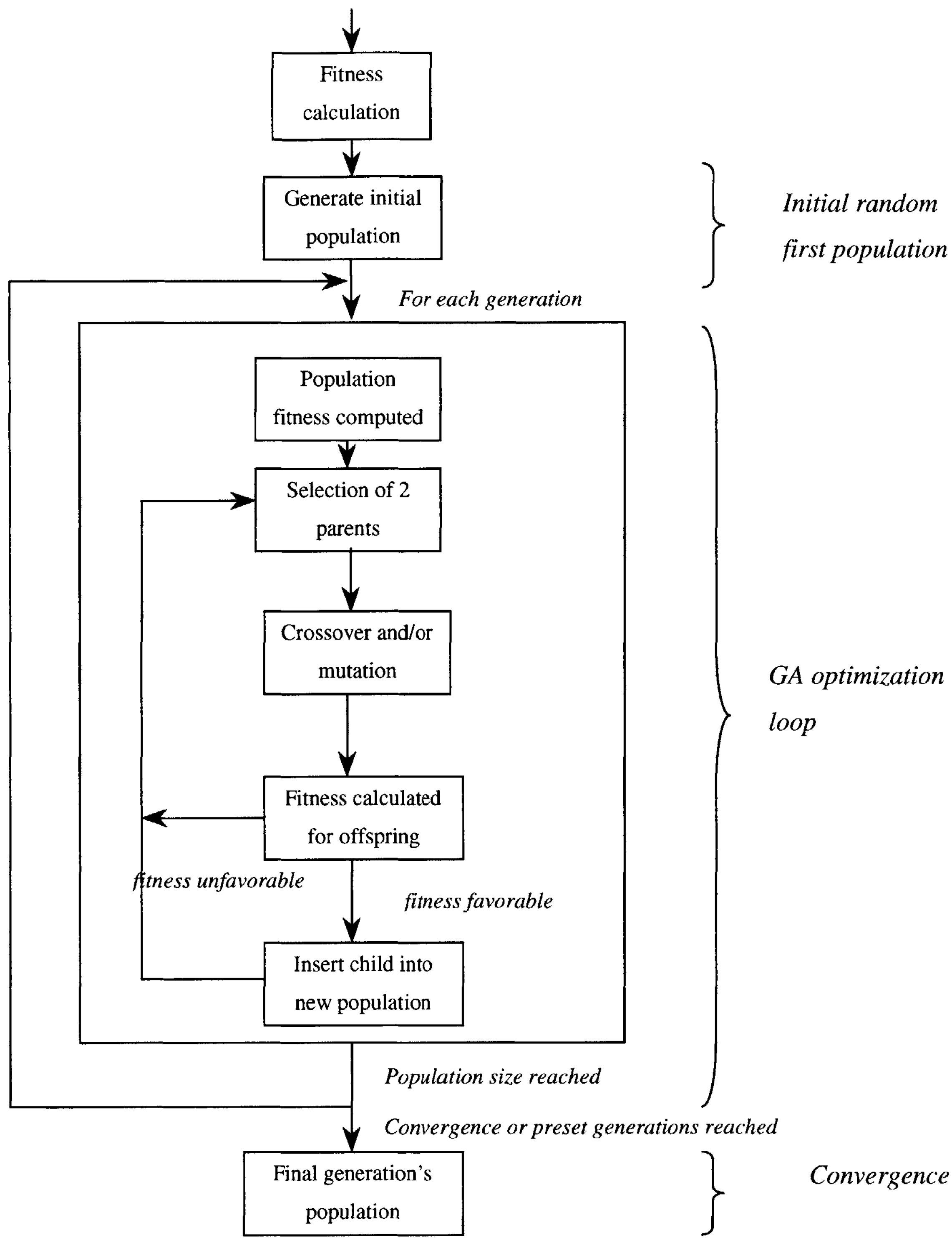


Fig. 2.1 Flowchart of how a simple GA works

2.1.1.3 Selection and reproduction

The fundamental way through which new individuals are produced in a GA is through the mating of fit parents to produce children. At the reproduction stage for a GA, parents are randomly selected according to a selection criterion biased towards a high fitness value. Several reproduction operators are then used to create the new child.

Crossover randomly selects one (single point) or more (multiple point) positions in the genome where the bits of parent chromosomes are exchanged. The resulting child will have a new fitness value based on the different input values for the genes. The child may or may not be introduced into the next generation (depending on its fitness and similarity measurements) but if so introduced, it replaces a less fit member of the population. As we use real encoding for our experiments, we apply the **BLX** crossover technique for continuous variables where a random real result is selected over a search space twice the size of the distance bounded by the two parent genes, centered about their mean. (Senin 1999b).

After crossover, **mutation** is applied to each child with a probability based on a random coin toss scaled according to the user-determined likelihood of mutation occurring. High rates of mutation have been shown to be not useful in optimization searches, tending to resemble a random search (Goldberg 1989 pp. 111). We use mutation rates of 0.01 for experiments in this thesis. Previous work using the Struggle GA with direct representations and operators has shown that optimizations are quite insensitive to the mutation rate.

2.1.1.4 Convergence

The cycle of selection and reproduction repeats itself until the termination criteria of either a preset number of generations or a certain measure of convergence is reached. The population would have ideally by then converged to several maxima located within the search space.

2.1.1.5 GA Statistics

Statistics used to measure the performance of the population are best-of-generation, online and offline performance scores (Davis 1987 pp. 67-68). Best-of-generation scores return the best score of the generation. Online scores measure the mean of all function evaluations up to that point in time and offline scores measure the mean of the best performance values for each generation up to a particular time. We will mostly use best-of-generation scores for this thesis.

Standard deviation of the population and diversity measures are also commonly used population statistics. Standard deviation is the square root of the sum variance of every individual in the population. It is given by :

$$\sigma = \sqrt{\sum_{1}^{T} (x - \bar{x})^2} \qquad \text{Eq. (2.1)}$$

The diversity of the population is the mean of the diversities of all the individuals in the population at a given generation. Direct encoding is used for the experiments in this thesis, so the distance measure between two individuals is simply the normalized euclidean distance (difference in real value) between them (Eq. 2.3). The diversity measure (similarity) between two genes is scaled according to the maximum possible distance between them (Eq. 2.2).

$$Distance\ (a, b) = |\ a - b\ | \qquad \text{Eq. (2.2)}$$

$$Similarity\ (a, b) = 1 - \frac{Distance\ (a, b)}{maxDistance} \qquad \text{Eq. (2.3)}$$

Therefore, identical genes have a similarity of one and maximally different genes have a similarity of zero.

2.1.1.6 Common GA problems & solutions

There are several unique problems that exist with overly dominant individuals in the population. In the use of ANNs within GAs, we are concerned with premature convergence and the maintenance of a large number of optimal peaks.

Premature convergence occurs when early on during the optimization, the existence of overly dominant individuals produces a population that very quickly converges. While this seem advantageous, these individuals may not be the globally optimal solution and this occurs at the expense of producing potential individuals that have a higher fitness in other areas of the search space. This can be prevented by using **scaling** techniques that reduce the dominant effect of these individuals early on in the optimization. We use linear scaling for our GA.

Niching techniques like **crowding** and **tournament selection** help to preserve the diversity of the population by having the child replace the most similar individual in the population. We use a variation of crowding called the **Struggle Algorithm** (Gruninger 1996) developed within the MIT CADLAB. The struggle algorithm has been shown to consistently locate a larger number of optimal peaks than other niching methods such as deterministic crowding and restricted tournament selection. This is suitable for use in a GA/ANN optimization as we want to maintain a large number of maxima to decrease the negative impact of neural net inaccuracies. The pseudocode of the struggle algorithm is as follows:

REPEAT until stop criterion is met :

1. Select parents P_1 and P_2 randomly (uniform selection)
2. Cross parents P_1 and P_2, yielding and offspring C'
3. Apply mutation and other operators, yielding C'
4. Find the most similar individual R, to C' in current population
5. If $f(C') > f(R)$ replace R wth C'

2.2 Neural Networks

Artificial Neural Networks (ANNs) are meta-models that use the principles of the workings of parallel data processing of neurons observed in a brain. Algorithms are fed training data and produce a system of neurons (nodes) that combine to simulate an output similar to that of the actual system. The concepts for neural models were first proposed by Frank Rosenblatt in 1958 (Masters 1993), but active research into neural networks only accelerated in the 1980s. It is now used in numerous applications where surrogate modeling is required.

The main advantage of neural networks over other meta-modeling techniques is that it does not require an explicitly specified model to work. Finding a deterministic model that fits closely with available data is often a problem for other meta-modeling techniques like regression. Neural networks have been shown to be effective for modeling design problems and shorten the time required to obtain a result given a large set of available existing data for CAD models and environmental design (Denis, 1998; Sousa, 1999).

For this thesis, we use a multiple input, single output, feed-forward neural network. The architecture of the network consists of 3 layers, with the hidden layer having 30 nodes. A back propagation training algorithm was used to train the neural network.

2.2.1 Review of how Neural Networks work

Fig. 2.2 shows in diagrammatic form how a simple neural net functions. The steps of adding training data and propagation of the training algorithm are outlined here before a review of each aspect is done.

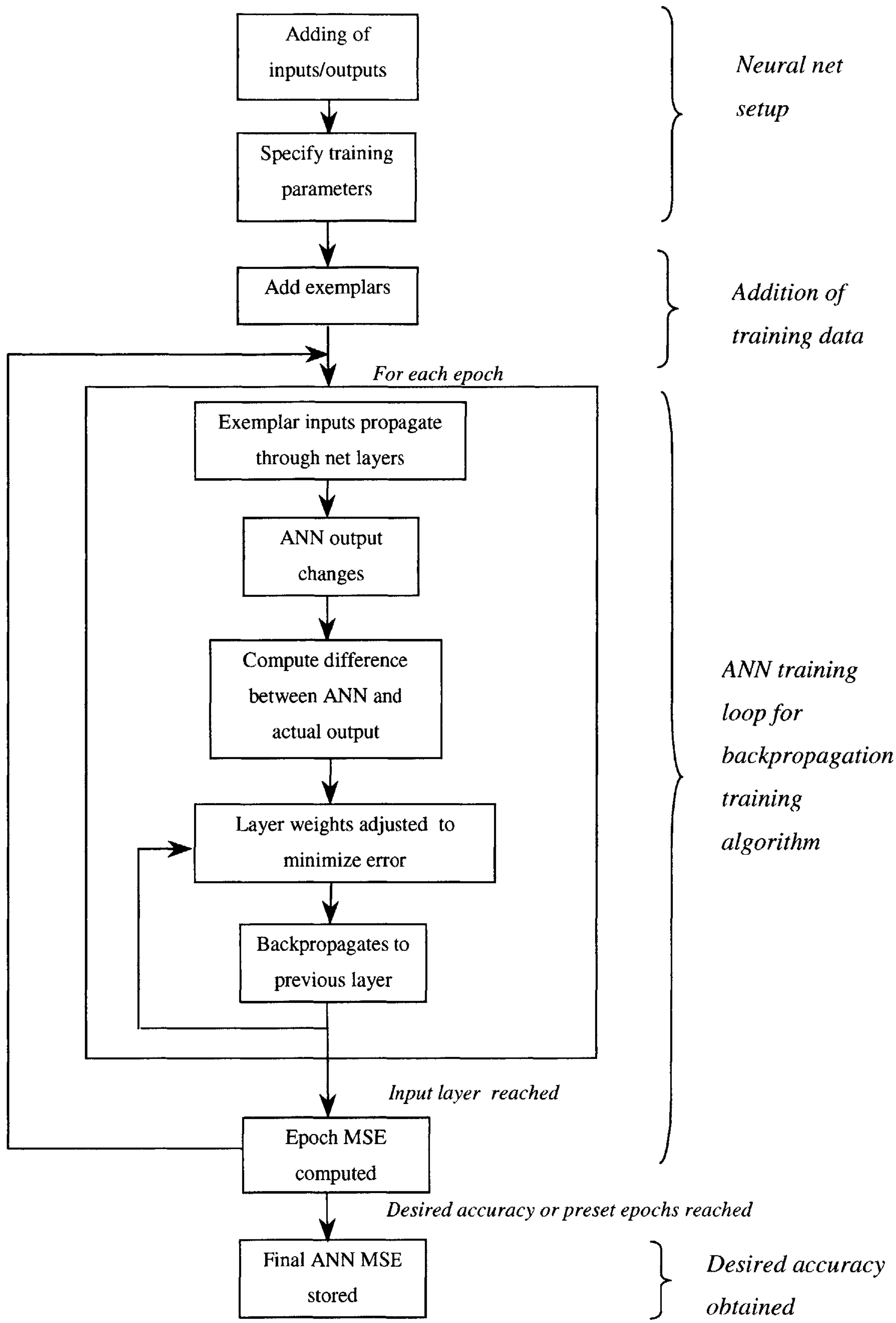


Fig. 2.2 Flowchart of how a back propagation network works

2.2.2 Network topology

One of the most popular and important form of neural networks used is the **feed forward** (FF) neural network (Schalkoff 1997). Feed forward nets commonly use the **back propagation,** also known as generalized delta rule (GDR), training algorithm.

The feed forward network generally consists of a set of three mutually exclusive layers of nodes. The **input layer** serves as a holding site for the inputs and the **output layer** contains the output nodes. Between these two layers lie the **hidden layer**. The hidden layer itself may have one or more actual layers. Between these layers of nodes lie links, or **weights**, which connect the nodes between the layers. In a feed forward network, each node in a lower level layer is connected to every node in the subsequent higher layer. The output value of a lower layer node is scaled proportionally by the weights to effect a change on the value of the higher layer node it is connected to (Fig. 2.3).

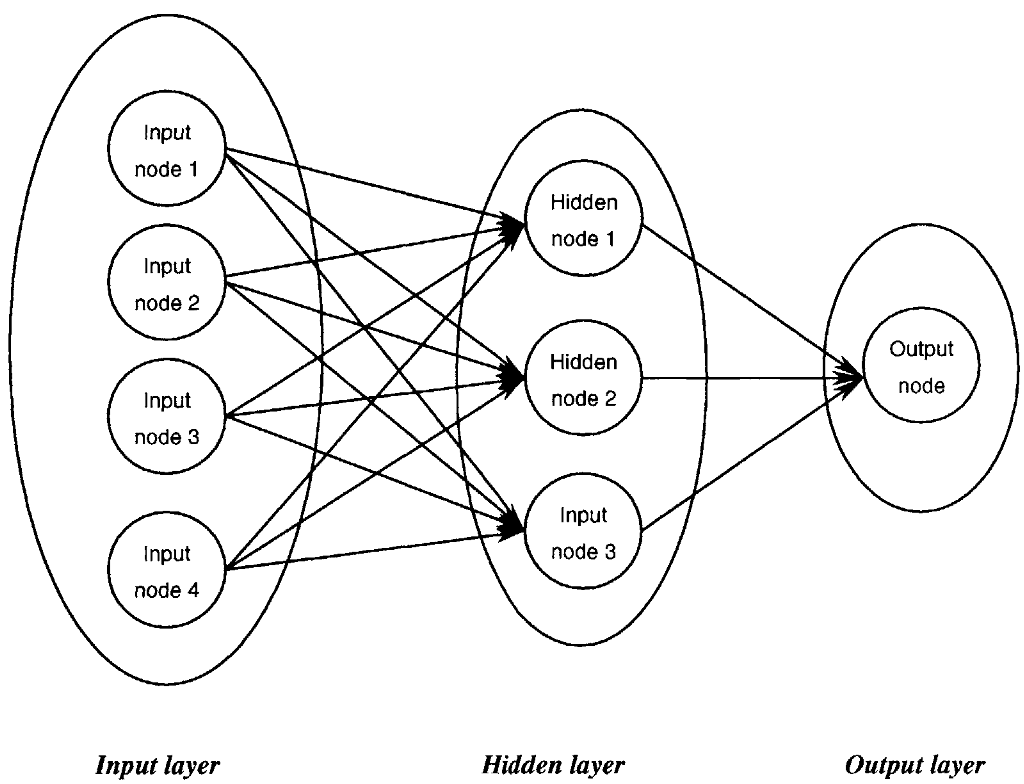


Figure 2.3 Structure of 3 layer feedforward neural network with 4 inputs, 1 output and 15 weights.

In the computation of the value of the hidden layer node, the dot product of all the inputs and their corresponding weights are first added to obtain an intermediate scalar, s :

$$s = \sum_{n=1}^{N} w_n . x_n \qquad \text{Eq. (2.4)}$$

This dot product is then passed through a nonlinear **activation function,** $f(s)$, which is normally the sigmoid function :

$$f(s) = \frac{1}{1+e^{-s}} \qquad \text{Eq. (2.5)}$$

In training the neural network, the inputs that are passed through the network change the values of the hidden layer nodes, which in turn modify the output. In general, an increase in the number of hidden nodes enables the neural net to model problems of increasing complexity. The increased number of nodes enable a greater fine-tuning as each one serves as a detector that contributes a small portion to the final output.

2.2.3 Training algorithms

Training algorithms are used to facilitate **learning** by the neural network. There are generally 2 types of learning. In **supervised learning**, training inputs with actual outputs are provided to the net and the output value of the net is compared to the actual output value for each data set. In **unsupervised learning**, the ANN organizes the knowledge into categories that appear to have similar characteristics or behavior. We use **back propagation** training algorithm which is based on supervised learning.

Back propagation is based on a deterministic approach in propagating the errors backward through the neural net and updating the weights in the process.

The algorithm minimizes the sum of output errors, E(w), over the entire training set :

$$E(w) = \frac{1}{p}\sum_{p=0}^{p-1} E_p \qquad \text{Eq. (2.6)}$$

p is the number of exemplars in the data set

E*p* gives the output error of the p*th* exemplar and is defined as :

$$E_p(w) = \frac{1}{2}\sum_{j=0}^{N-1} (T_{pj} - O_{pj}(w))^2 \qquad \text{Eq. (2.7)}$$

O_{pj} and T_{pj} are the network and target outputs respectively of the jth output node.

After computation of the error, the weights to the previous layer are modified according to the rule :

$$w \longleftarrow w - \mu \nabla E(w) \qquad \text{Eq. (2.8)}$$

μ = the learning rate.

The modification of the new weights is proportional to the negative derivative of the error. This is proportional to μ, the **training rate.** A large training rate can cause large fluctuations in weight changes, while a small training rate slows down the training excessively. We use a training rate of 0.003 for our neural nets.

The number of training exemplars required for a neural network is generally at least 5 – 10 times the number of nodes in the network. The common measurement for the accuracy of a neural network is the **Mean Squared Error** (Rogers 1992 pp. 58) :

$$MSE = \sum_{i=1}^{N} (t \arg et - output)_i^{\,2} \qquad \text{Eq. (2.9)}$$

2.3 DOME Implementation

The use context of genetic algorithms and neural networks for this thesis is the optimization of large-scale distributed design models. In a computer-based **design model**, many **design variables** interact with each other in deterministic relationships to produce various design solutions. These design solutions are measured in terms of performance metrics or **performance variables** (Senin, Wallace and Borland, 1997 pp. 2). Within a design model, problems are often decomposed into sub-problems to manage the complexity of a modeling task.

The overall framework, in which genetic algorithms serve as an optimization tool and neural nets as meta-models, is called **DOME** (Distributed Object-based Modeling Environment) (Wallace et. al. 1999). DOME is a distributed modeling environment for integrated modeling developed at the MIT CADLAB. It allows independent users using their own preferred modeling tools to interact dynamically with each other through the internet using its framework. In this environment, designers build their own models as object entities and are able to use dependencies based on their own internal as well as external objects. Similarly, these models are able to provide dependencies to external models that propagate throughout the entire environment in real time – a truly distributed collaborative engineering system with the ability run a virtual engineering corporation.

Within this framework is the genetic optimization module we use that enables design parameters to be entered as search variables and the neural net module which simulates subsystems by providing an output when given pre-defined inputs. Various methods have been used in the search for an optimal solution to such design problems (Senin, Wallace and Borland, 1997 pp. 5), but genetic algorithms have proven to be especially effective in due to their robustness in exploration and exploitation of the search space (Senin, Wallace and Borland, 1999). Neural networks have also shown to be useful in reducing the amount of time required to obtain results from CAD and environmental models within this environment.

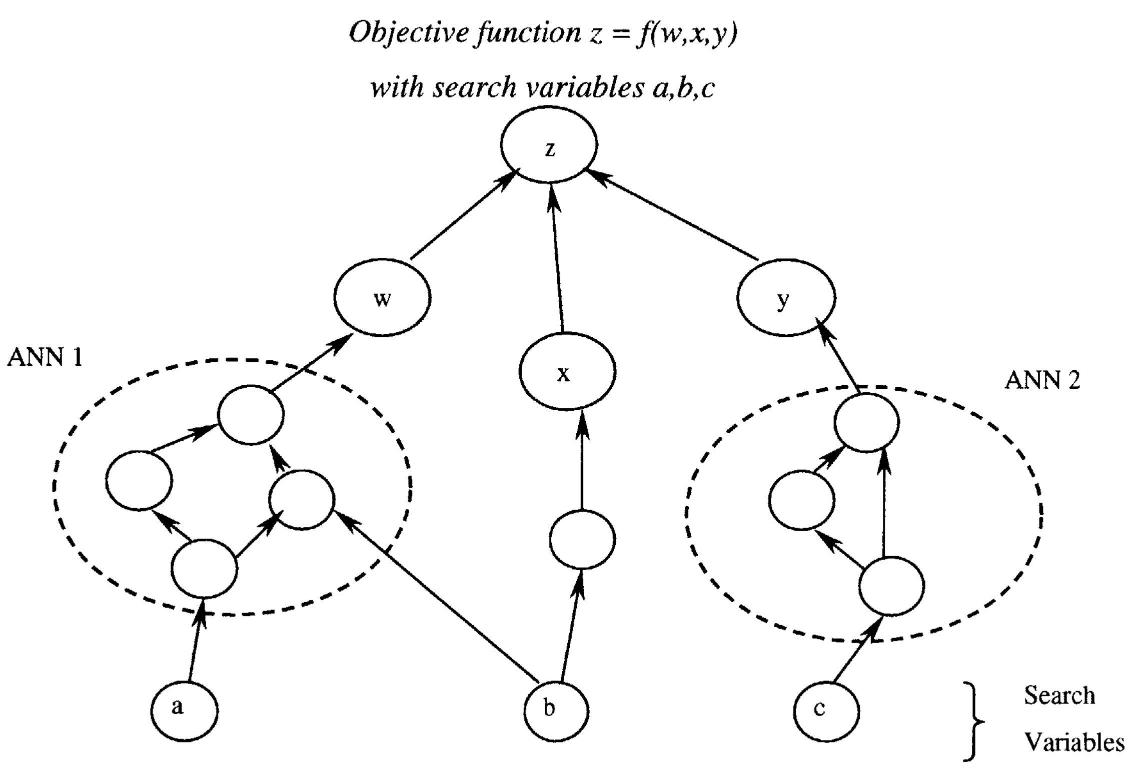


Fig. 2.4 Schematic of real and neural net combinations for an optimization.

3. Neural Net use in Genetic Algorithms

In using neural nets to substitute for computationally expensive sub-problems in an optimization, we need to know the accuracy of the results as compared to those obtained by a traditional optimization. How good are the final results using an ANN as compared to that for a normal optimization? Also, to what degree of accuracy do we need to train the neural net? We do not want to over train the net, which will take more time, but neither do we want to under train the net, which would provide poor modeling results. What factors will affect our decisions of what and when to use neural nets during a GA optimization? These are questions that will be addressed in this chapter.

First, we will analyze ANN use according to four factors of ANN accuracy, convergence, time and sensitivity. A framework with which we will relate the accuracy of the results to the accuracy of the neural net will be presented. The neural nets are divided into 3 main categories that define their use in an optimization in Section 3.2. The bounds of these categories are computed with respect to the accuracy of the neural nets (using Mean Squared Error). A relationship is also obtained to compute the point during the optimization to change over from using the neural net to the actual function evaluation for nets that do not have such a high degree of accuracy. The use of statistics indicative of convergence in a GA population are also discussed with context to their effectiveness in dictating a point to turn off less accurate neural nets in Section 3.3. The use of function evaluation time is also discussed and methods to promote optimal use of multiple neural nets with different service times are proposed. Finally, some background is provided on how sensitivity analysis of the different contributions of the search variables to the objective function can be utilized in deciding the length of use time for the neural nets.

3.1 Description of use and factors

The use of ANNs for GA optimization is not an exclusive one. The GA will have a mixture of search variables that consist of both actual function evaluations as well ANN models. This is reflected in the choice of search variables and the formulation of the objective function. A simple diagram showing the usage of 1 ANN and 1 actual design variable in a GA optimization is shown in Fig. 3.1. Such a choice is left to the discretion of the expert user.

GA objective function $z = f(x, y)$

(using search variables a, b, c)

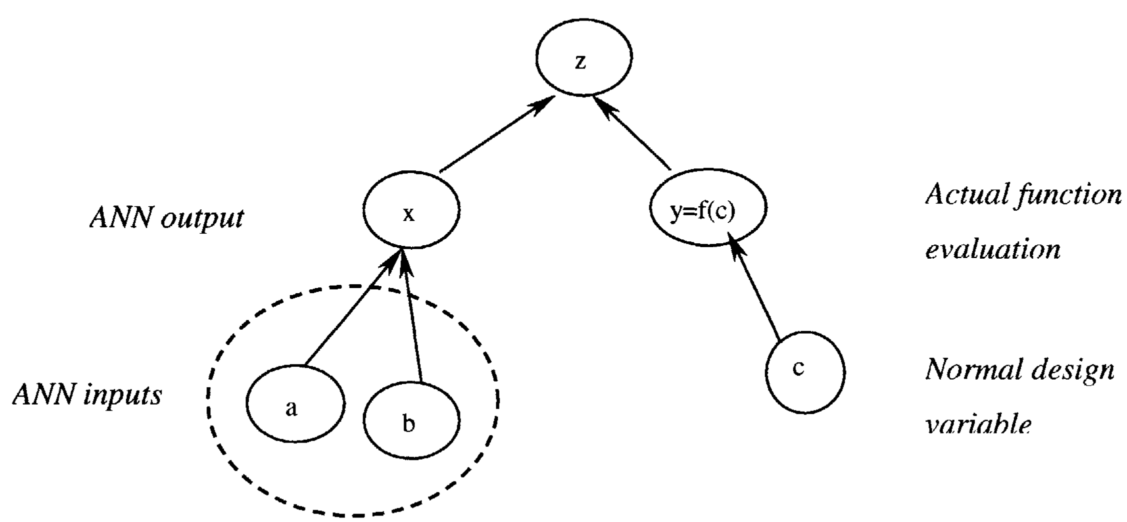


Fig. 3.1 Usage of ANNs in an optimization

The primary purpose of neural nets usage in genetic algorithms is to provide a faster way of obtaining results that are close to that obtained by a normal GA search. The decision of when to use the neural nets are determined primarily by 4 factors :

1. ANN accuracy
2. Convergence
3. Time
4. Sensitivity

3.2 ANN accuracy

3.2.1 Classifications of ANNs used for GAs

For our purposes of using ANNs in a genetic optimization, we classify them into 3 qualitative categories :

Very accurate ANNs	Partially-accurate ANNs	Inaccurate ANNs
Can be used for the entire optimization	Are used to some degree. The GA switches over at some point	Are not used at all

Fig 3.2 Classification of ANNs

We use the Mean-Squared-Error (MSE) to define the accuracy of a neural net (Eq. 2.9). We also need to define the **upper bound** (MSE below which a ANN can be used 100% of the time), the **lower bound** (MSE above which the ANN should not be used at all), and the **changeover region** where the ANN can be used for a period of time before being turned off. These three regions allow us to assign quantitative values to demarcate them and provide an idea of expected performance of the net during a hybrid optimization.

3.2.1.1 Determining the 3 ANN boundaries

With this definition of 3 distinct categories, we can proceed to define the boundaries for each region. To do so, we first define the measurement criteria that an ANN needs to achieve to be classified under each category. This measurement criteria relates the performance of the GA/ANN optimization to that of a normal optimization.

Measurement criterion

There are several possible ways of defining a measurement criterion for each of the 3 boundaries. The most straightforward method would be to simply use the MSE of an ANN without regard to its optimization performance. For example, we could say that if we have a MSE of 0.002 between an ANN and its function, we can define this region to be very accurate arbitrarily. However, with such a definition, we are unable to relate such a criterion to provide us an idea of how close the GA/ANN result is to that for a normal GA optimization.

A more useful measurement method would be to fix the desired performance of the GA/ANN with respect to a normal GA optimization, and observe the required accuracy of the neural net for this to occur. For example, if a GA/ANN with a MSE of 0.002 (used for the entire optimization) gave a final optimal score that matched that of a normal GA to within a certain percentage, we would define it as being very accurate. On the other hand, if a GA/ANN with a MSE of 0.01 gave a final optimal score that deviated from that of a normal GA beyond a certain percentage, we would define such an ANN as being inaccurate.

Keeping in mind that training an ANN beyond a certain degree of accuracy requires an exponential increase in time, we do not want to seek too exact and strict a requirement. Again, we are not looking for a ANN to totally substitute a normal run and are not seeking to use ANNs on an all-or-nothing basis. Rather, our objective is to create a population of individuals that is converged around promising peaks, hopefully including the global. If we are able to use the neural net for the entire optimization period, we would most certainly want to, given the amount of time savings this would provide. However, if this cannot be achieved, it would still be useful to use the net to help us achieve a relatively healthy population quickly before using actual computation to exact a more rigorous search.

Upper Bound

The method of defining the upper bound (very accurate ANN) is shown here (Fig. 3.3) :

A very accurate ANN gives a final optimum that is within 1% of the final optimum of a normal GA run.

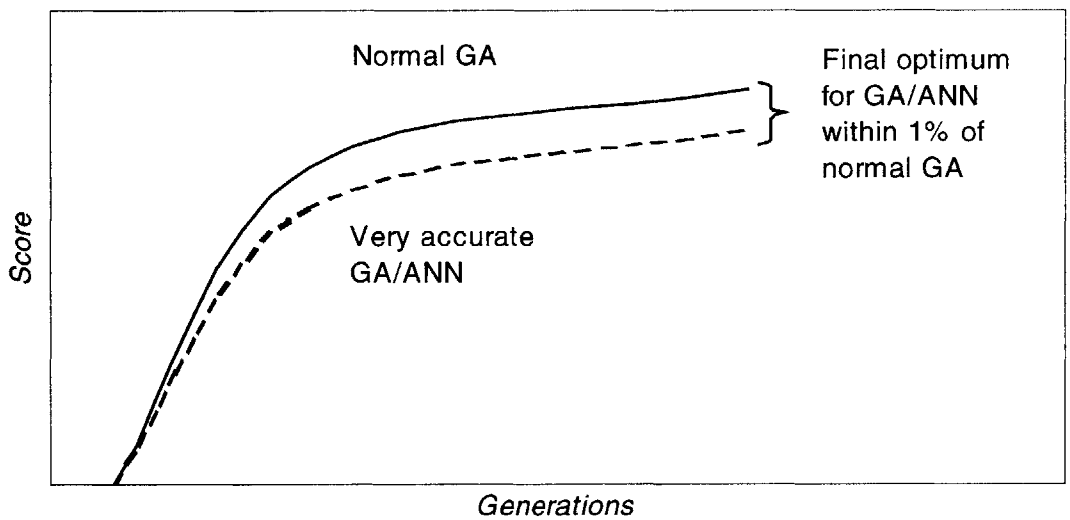


Fig. 3.3. Upper bound : 1% deviation from final optimum

This means that when the GA uses a very accurate ANN (prespecified MSE), we can be confident of obtaining results that are within 1% of that obtained by a normal GA run with similar population size and generations (Fig. 3.3). Correspondingly, all ANNs with

MSEs smaller than the upper bound (more accurate) can be expected to give results with deviations smaller than 1%.

Lower Bound

In deciding the lower bound, what is acceptable for use at the lowest level is arbitrary. However, we find that inaccurate ANNs used for optimization produce extremely poor results (Chapter 4.2.1). The resulting population is converged on the wrong peaks and the GA is unable to recover even after turning off the neural net and further running the optimization using actual function evaluations (due to excessive loss of genetic diversity). Clearly there needs to be a bound to prevent such mistaken usage from occurring. The idea of having the lower bound is to provide a boundary measurement that prevents usage of ANNs that produce a performance that deviates beyond a certain amount. We want to produces a population that has individuals on promising local peaks, but without excessive convergence on and overexploitation of any one of them. This will present a set of promising potential individuals very quickly for the genetic algorithm to exploit without running the actual GA in the initial few generations.

In defining the lower bound, we require the definition of both the changeover position as well as the accuracy.

We define the lower position at 10% of the number of generations. This is arbitrarily chosen, but is deemed to provide sufficient time for the GA to fully utilize the ability of the neural net. For the comparison of scores, we do not want to use the final optimal score for comparison, but rather the optimal score at the 10% generation point. This is to prevent the possibility of an excessively positive bias for the GA/ANN optimization that could occur since 90% of the optimization will be run using actual function evaluations.

The method of defining the lower bound (inaccurate ANN) is shown here (Fig. 3.4) :

An inaccurate ANN gives a 10% generation optimum that deviates more than 1% from a similar score for a normal GA run. The lower bound is the MSE of the first inaccurate ANN.

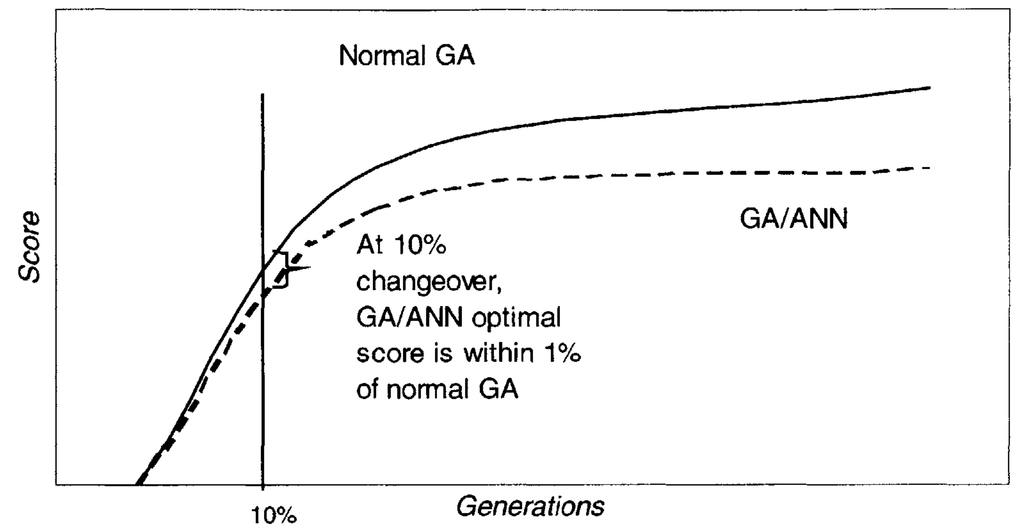


Fig. 3.4 Lower bound -1% deviation at 10% generations

For example, for an optimization with 100 generations, a GA using an inaccurate ANN would have a optimal score at the 10th generation that deviates by more than 1% from the optimal GA score at the 10th generation.

3.2.2 Determining the changeover point

Having the bounds defined in the previous sections, we want to focus on addressing the changeover region in more detail. An ANN in this region will be used from 10% to 100% of the optimization period depending upon how accurate (MSE measure) it is. How do we decide when to changeover for partially-accurate nets in a GA/ANN optimization?

There are several possibilities that can be considered. The diversity of the population (too low could mean loss of genetic diversity), the standard deviation (to narrow a deviation is unhealthy), measurement of the changeover score (compared to the globally known score) are potential indicators for changeover. If these parameters exhibit relationships that show a close correlation to a desirable GA/ANN optimization, we would have found heuristics to suggest changeover points in using partially-accurate ANNS for a GA optimization.

Another possibility would be to use a linear fit between the lower and upper bounds if no GA parameters are found to be useful in determining when to changeover (Fig. 3.5). This linear relationship would relate the changeover point to the accuracy of the neural net in question.

The changeover point in this case would predict that a GA/ANN that had a changeover at the designated generation would produce a changeover result that was within 1% of that obtained by a normal GA optimization at that generation. This would not only provide us with a heuristic for changeover, but might also provide us with a heuristic indication as to the accuracy of the optimization.

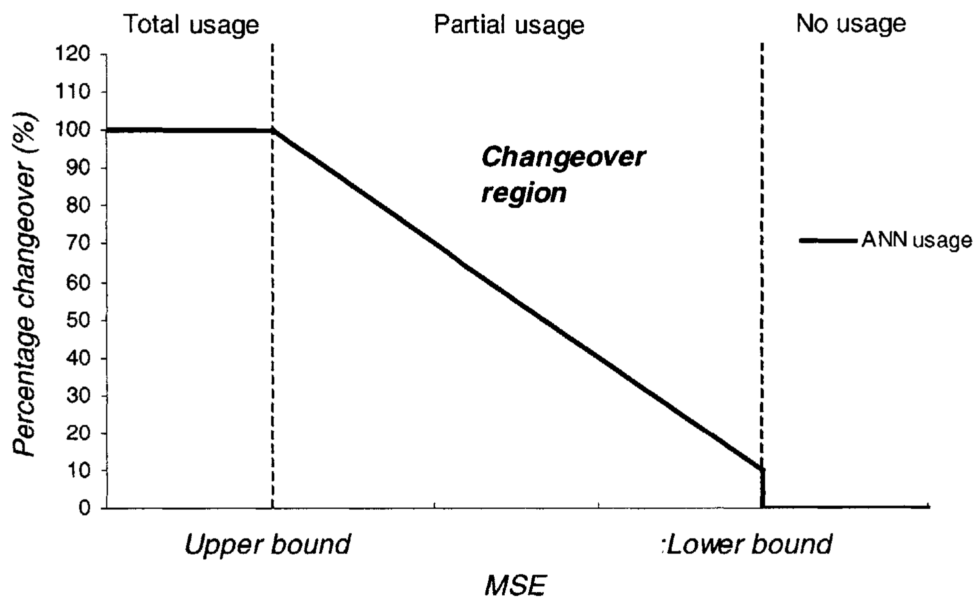


Fig. 3.5 3 regions of ANN usage based on its accuracy

3.3 Convergence

As the optimization progresses, the standard deviation and diversity of the population should decrease as the individuals converge towards either local or global peaks. Such a convergence could point to a suitable changeover time for a partially accurate ANN. We want to investigate the possibility of these two statistics giving indications as to when to switch the population from using neural nets to the actual computation.

Standard deviation is the root of the sum total variance of every individual in the population (see Chapter 2.1.1.4 for more detailed explanation). We want to look at standard deviation measures of GA/ANN optimizations of varying accuracy to find trends that could relate the measure to suggest a changeover position. We also want to run optimizations with various changeover points for the same ANN to see how the standard deviation changes during and after changeover.

Diversity is the measure of how different the individuals are to the rest of the population (see Chapter 2.1.1.4 for more detailed explanation). Generally, we would not want a population that is too similar as this reduces the number of potentially global peaks we would have after changeover. We also run experiments similar to that for standard deviation.

3.4 Service Time

The decision as to whether to utilize an artificial network is based upon how long it takes to evaluate a search variable/s change. If the change takes a significantly long time to propagate, then it becomes useful to replace that search variable/s with the ANN model (depending on its accuracy).

It is proposed in this thesis that service time be factored in only after all the ANNs available for an optimization are in use. In a large scale model, all ANN inputs that are needed should be brought in as search variables into the Genetic Algorithm and heuristics be used to decide based on ANN parameters (ANN error, convergence, sensitivity) how long to run each ANN for, if at all. It is not proposed that time be a factor that is used in deciding changeover.

The **Service Time** of an ANN (for purposes of the GA) is the total time taken for every ANN search variable input to individually affect the output. This will be the time that the GA will take to compute the objective function for each search variable input change.

3.4.1 Dominance

Service time will only be brought into consideration when the heuristics have switched off the first ANN (due to error or convergence). In parallel computation of services, we want to avoid the scenario where a slow service is already running, but a faster service is still using an ANN. This can be done with the concept of **dominance**. If a faster service using an ANN is already dominated by the existence of a slower service without an ANN, the faster service should switch over from using an ANN to using actual computation (Fig 3.6).

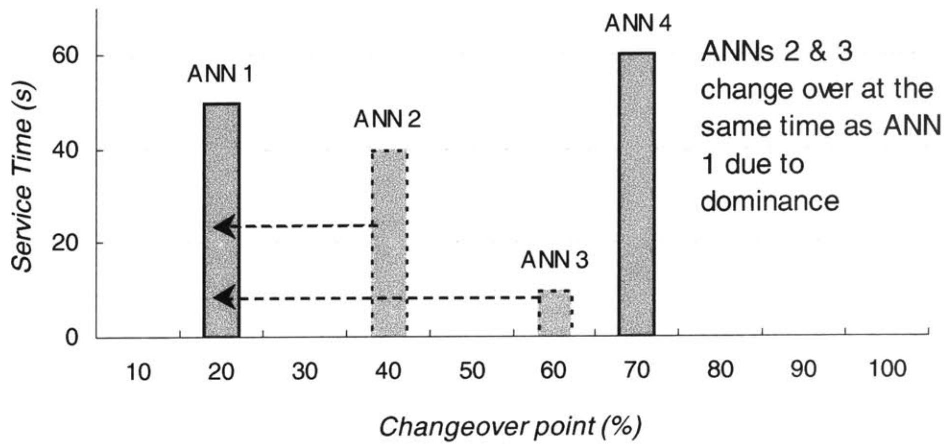


Figure 3.6 Changeover due to dominance

3.4.2 Time creep

The second way to integrate Service time is through **time creep**. An ANN whose service time is only a second slower than a non-ANN service should changeover at some time in the near future as the differential increase in time is minute. To enable this, it is proposed that a time creep be implemented. Once an ANN has undergone changeover, a linear time creep is started. This creep will move towards services slower than the particular ANN at a calculated rate specified by the user, and will cause all slower ANNs whom it catches up with to changeover (Fig 3.7). Once another ANN has undergone changeover due to reasons other than changeover creep, the changeover creep calculation will restart again with the new ANN.

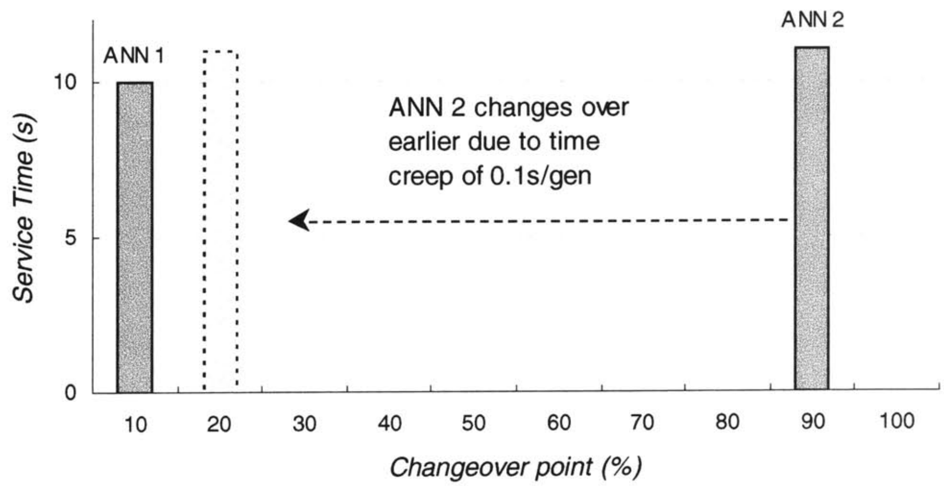


Fig 3.7 Changeover due to time creep

Both the dominance and time creep concepts assume that all other normal GA services compute instantaneously. For a more complex model where some non-ANN services are slow, dominance and creep will have to be extended to consider these services too to prevent the ANNs from being dominated by these slow services. However, it is not within the scope of this thesis to consider this.

3.5 Sensitivity Analysis

3.5.1 Description

Fitness sensitivity is important in the analysis as it allows us to make a more discerning choice in using a ANN which does not affect the score as much (perhaps in a particular region or globally), but has a prohibitively slow time. In such a case, in the interests of saving time, we may choose to use the ANN for a longer time than usual. In another scenario, we would want to train an ANN, which the objective function is very sensitive to, to a higher degree of accuracy.

It is not within the scope of this thesis to experiment into the details of sensitivity analysis, but a brief review of the possible methods available would be useful.

A review of sensitivity analysis literature reveals several common techniques :

1. Linear programming
2. Nonlinear programming (Lagrange multipliers)
3. Design of experiments (analysis of variance approach)

We take note that for the calculation of sensitivity analysis for a genetic algorithm (a computer model), many of these models attempt to account for noise in the model. Noise is not present however, in deterministically determined models as all factors contributing to the objective function are known (Simpson T 1997 pp. 8).

We will analyze each of these in turn by comparing the extent to which each approach meets our needs to determining factor sensitivity and computational effort.

3.5.2 Review of possible methods

1. Linear programming

Linear programming is an established optimization method performed on linear functions. Linear deterministic equations are used to constrain the search boundary. The most common method is to use the simplex method (a local hill-climbing technique) to obtain a local optimal solution.

Sensitivity analysis computations are derived from the **shadow price** of the factors. The shadow price gives the change in the objective function due to a unit change in a factor within the specified bounds, given that all other factors remain constant (Ignizio J., pp 219).

The main drawback to linear programming is that it does not account for higher order interactions (eg. x^2, xy) within and between the factors. It also requires deterministic equations linking the objective function to the factors to obtain certain required matrices, which is not possible under certain search contexts (e.g. Finite Element Methods).

2. Nonlinear programming

Nonlinear methods use deterministic equations (including nonlinear equations) to constrain the search boundaries. The **Lagrange multipliers** technique is utilized to find the optimum solution and the multipliers also provide for sensitivity analysis (Lewis, R. pp 15).

While nonlinear programming enables higher order interactions to be accounted for, it still requires deterministic equations that relate the variables and objective function and is not feasible for our type of problem.

3. Design of Experiments approach

A **Design of Experiments (DOE)** approach identifies how certain inputs affect the output in a process. It takes samples of combinations of inputs, and tries to separate the real signal from the noise to determine if the factors are statistically significant.

In a deterministic computer-designed model which DOME is, all changes are significant if they affect the output. There is no noise present as changes are fired as a direct result of an input change. Various DOEs are compared in Fig. 3.8 with their corresponding number of require data sets in Fig. 3.9.

Design	**Objective**	**Advantages**	**Disadvantages**
3^k full factorial	• Estimates all linear & quadratic effects	All main effects and interactions accounted for.	Large computation costs (2^k)
^{3k} fractional factorial	• As in 3k full factorial • Used when interactions are few	Fewer computations than full factorial	Limited interactions. All right according to sparsity of effects principle.
D-optimal	Minimizes non-orthogonality	Allows us to specify time constraints (by limiting no. of runs)	May not be as accurate

Fig. 3.8 Summary of 3-Level Design types (Schmidt, S. pp 3-34)

Design	K = 3 with no interactions	K = 4 with 3 2-way linear interactions	K = 5 with 1 2-way linear interactions	K = 5 with all 2-way linear interactions	K = 7 with 7 2-way linear interactions
3k full factorial	27	81	243	243	2187
3k fractional factorial	9	27	27	81	81
Box-Behnken (B-B)	15	27	46	46	62
Central Composite Design (CCD)	13	27	46	46	62
D-optimal	7	11	12	21	22

Fig. 3.9 Comparison of Design Types for Quantitative Factor Scenarios

(Schmidt 1997 pp 3-33)

Of the options for a Design of Experiments approach, the D-optimal design requires 1/3 of the computation cost of the second least expensive DOE method (Table 3.24). It is flexible to accommodate different type of interactions (both linear and quadratic). Its weakness is its trade-off in inaccuracy. It would seem that a D-optimal design would require fewer data points as compared to other DOE methods. However, it proved to be inaccurate even for simple models. The computation of sensitivities is a highly complex issues given the large landscape and contributions of multiple variables. Perhaps combinations of DOE and other techniques may be explored and prove more effective. A more thorough experimentation on the computation of sensitivities is beyond the scope of this thesis, though knowledge of the sensitivities would prove useful in the future for extending the usage of ANNs that have a long service time while only having a small effect on GA the objective function.

4. Experiments & results

In this section, experiments were run to test the hypotheses reviewed in Chapter 3. Tests were run using C++ code based on GA and ANN libraries developed within the MIT CADLAB. Two test functions, the Himmelbau and the Hilly, were utilized both individually and in combination for the GA optimization.

Details of the code and hardware setup for the experiment are first covered in section 4.1. A range of GA/ANN optimizations were then done on both functions for ANNs of carefully selected accuracy at regularly-spaced changeover points in section 4.2.1 to provide a good initial overview of the range of optimization results for the various combinations of ANNs and changeover points. Tests were done in the next 3 sections (4.2.2 to 4.2.4) to obtain the lower and upper bounds before combining them to obtain the changeover region. The changeover region was also tested for accuracy by validating data points (including those not used for calibration) onto the region.

Section 4.3 investigates the usage of convergence and population statistics including standard deviation and diversity as potential measures for changeover points. The GA parameter of population size was next investigated in section 4.4 and shown to be promising in providing a healthier population during changeover, resulting in higher final optimization scores.

4.1 Experimental setup

The genetic algorithm was based on C++ code written within the MIT CADLAB. Many thanks to Matt Wall, whose GALib library served as the base code and also to Nicola Senin, who adapted the GALib for DOME use. The Struggle Algorithm was used for all tests with a mutation rate of 0.01, single crossover rate of 1.0 and direct encoding with BLX crossover. The average of 10 runs was used to obtain each result.

The artificial neural network used was based on C++ code written within the MIT CADLAB for the DOME system by Juan Deniz. A multiple input, single output, feed-forward ANN with 30 hidden layers and back propagation training was used. A training rate of 0.003 with a momentum of 0.9 which are common values for ANN training.

Additional code was written to integrate both libraries together to enable the required experiments to be conducted.

The hardware used in these experiments consisted of a 300 MHz Pentium II with 128 MB of RAM.

Two test functions were used for the experiments. A relatively smooth function known as the Himmelbau function (Fig. 4.1) and a more complex function called the Hilly (Fig. 4.2) function.

$$Himmelbau(x, y) = 5 - \frac{(x^2 + y - 11)^2 + (x + y^2 - 7)^2}{200} \qquad \text{Eq. (4.1)}$$

for $x, y \in [-5; 5]$

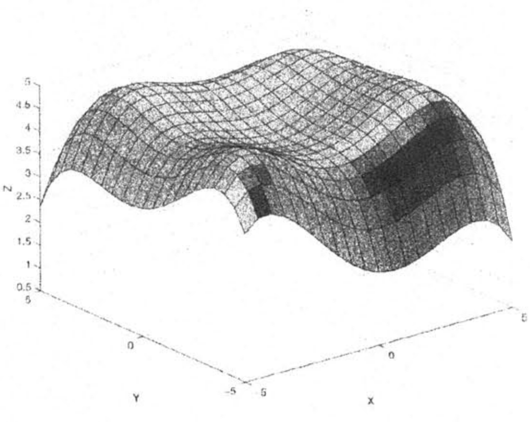


Fig 4.1 Plot of Himmelbau function with global optima of 5 at (3,2)

$$Hilly(x,y) = 10*[e^{-\frac{|x|}{50}}(1-\cos(\frac{6}{100^{\frac{3}{4}}}\pi|x|^{\frac{3}{4}})) + e^{-\frac{|y|}{250}}(1-\cos(\frac{6}{100^{\frac{3}{4}}}\pi|y|^{\frac{3}{4}}))$$

$$+2*(e^{-\frac{(b-x)^2+(b-y)^2}{50}})] \qquad \text{Eq. (4.2)}$$

$$with\ b = (\frac{5}{6}*100^{\frac{3}{4}})^{\frac{4}{3}}$$

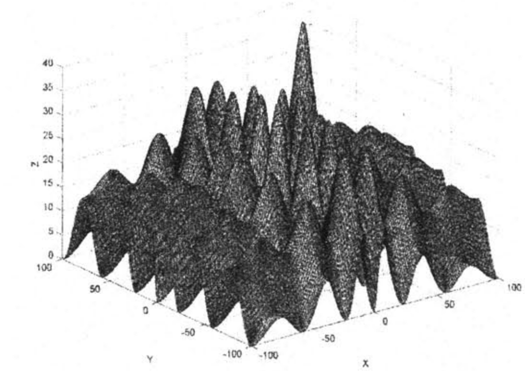


Fig 4.2 Plot of Hilly function with global optima of 38.78 at (b,b)

4.2 GA/ANN framework test

An overview of the range of optimization results of a GA/ANN was first obtained. Hybrid optimizations were done for different ANNs at a range of changeover points from 10% to 100% and compared to results of a normal optimization. Selected ANNs are shown in Fig. 4.3 along with details of the number of training points, training time and MSE and the genetic algorithm parameters. 182 and 1122 data points for the Himmelbau and Hilly function respectively uniformly mapped over the search space were used for training as they were shown to provide good modeling (Deniz 1998).

	Neural Network			GA parameters	
	Data points	Time (s)	MSE	Population size	Generations
Himmelbau	182	6351	0.0001	5	30
		1817	0.0005		
		1725	0.001		
		547	0.003		
		68	0.006		
		47	0.01		
Hilly	1122	103 758	0.003	30	150
		25 733	0.006		
		14 584	0.01		

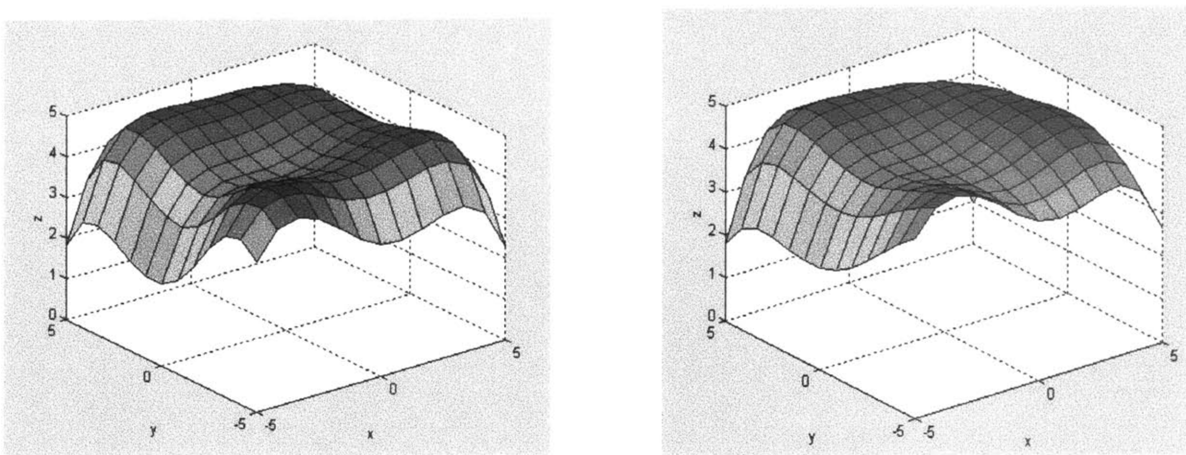
Fig. 4.3 Training and GA Parameters for selected GA/ANNs

For each ANN used, 11 GA/ANN runs were made with changeover points from 0% to 100% at 10% intervals over the entire optimization run. For example, with a ANN of

MSE 0.003 and GA generation of 100, we would first run the GA without using the ANN (0%). We would then repeat the process but instead now use the ANN for the first 10% of the run (up to generation 10) and use a normal optimization for the remaining 90% of the run. This process is repeated in intervals of 10% until we reach the full ANN utilization for the entire run (100%). The scores are then output into a file along with population statistics of standard deviation and diversity.

From Fig 4.3, the Himmelbau function clearly took much less time to train than the Hilly function. The Hilly function is a more complex function (compare Fig. 4.1 and 4.2) and the net required more data points and a much longer learning period. Additional research in obtaining a more accurate neural net by optimal data point selections and modification of the parameters in the future could probably reduce this training period significantly. For the Hilly function, a MSE accuracy of beyond 0.003 (took 28 hrs) was not possible as the exponential increase in time for additional accuracy made training prohibitive.

Surface maps of the functions are seen in Fig. 4.4 for the Himmelbau and 4.5 for the Hilly function. The degraded surface plot is clearly seen when the accuracy of the net is decreased from an MSE of 0.0001 to 0.0018. The optimal peak is not mapped properly for the net of lower accuracy. The Hilly function was plotted at an accuracy of 0.004 and already, it can be seen that the optimal peak was missed at this level of accuracy.



(a) (b)

Figure 4.4 Surface plots of the Himmelbau at (a) MSE 0.0001 and (b) MSE 0.0018

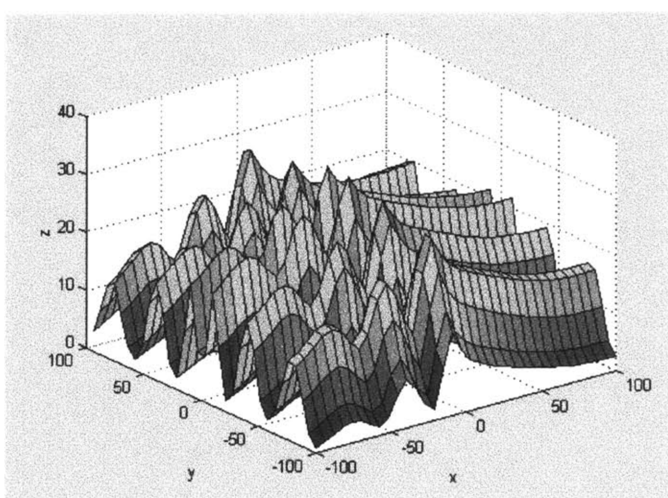


Fig. 4.5 Surface plot of Hilly function at MSE 0.004

4.2.1 GA/ANN optimization results overview

The hybrid optimization of various ANNs at differing accuracy and changeover points showed the degradation of final best-of-generation results as the mean-squared error for the nets increased. This is clearly seen in Fig. 4.6. However, we find that there exists a certain region of MSE beyond which the results at varying changeover points produce similarly accurate final best-of-generation scores. An example is shown in Fig. 4.7 of such an Himmelbau net with an accuracy of 0.0005. As the accuracy of the nets decrease, and the MSE increases, the final best-of-generation score starts decreasing as we use the net for a longer amount of optimization time (up to 100%). This is seen in Fig. 4.8. A further explanation is given after each figure :

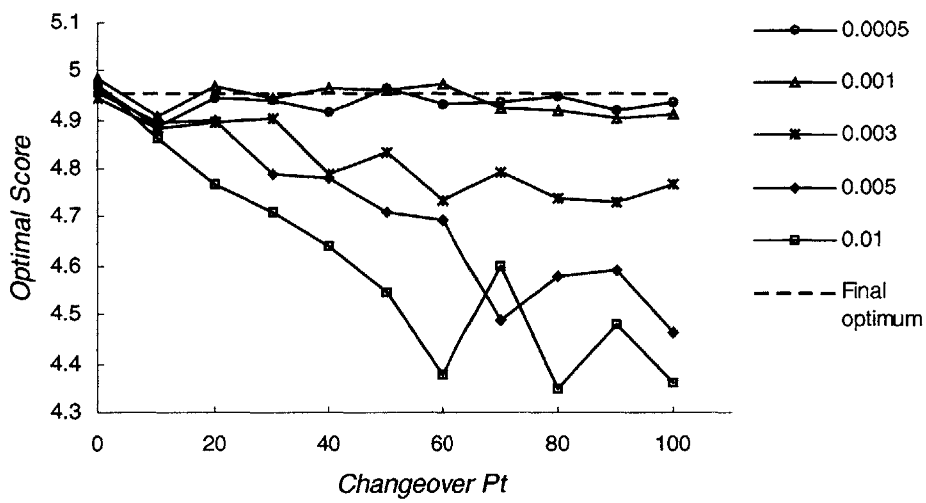


Figure 4.6 Plot of final score of various ANNs at different changeover points

Fig. 4.6 shows the final best-of-generation results for 55 different runs (11 for each net) done over different changeover points. The final optimal score of a normal GA without nets was 4.96 for the Himmelbau (shown by a dashed line). The net with an accuracy of 0.0005 had all its final scores within 1% of the final optimal on all changeover points (also in Fig. 4.7), while the net with an accuracy of 0.01 had scores that degraded to less than 4.4 when the net was used for the whole optimization.

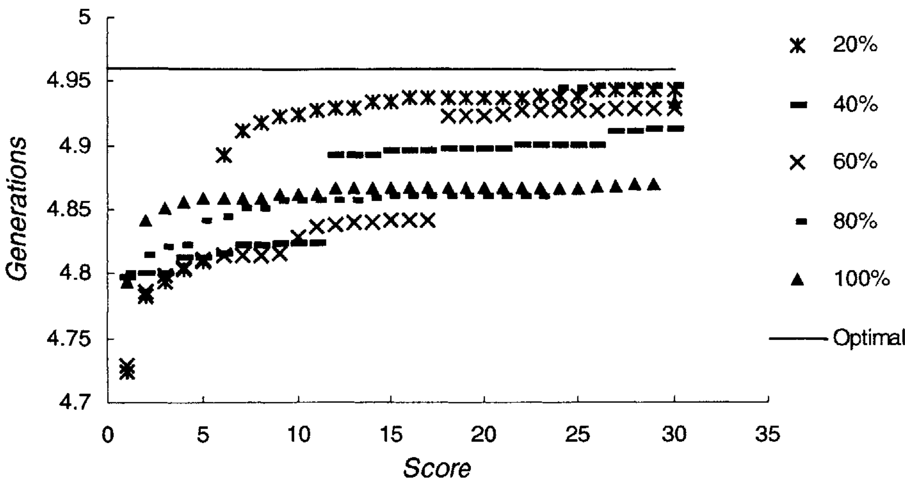


Figure 4.7 Plot of all best-of-generation scores for GA/ANN runs at varied changeover points for a Himmelbau ANN of MSE 0.0005

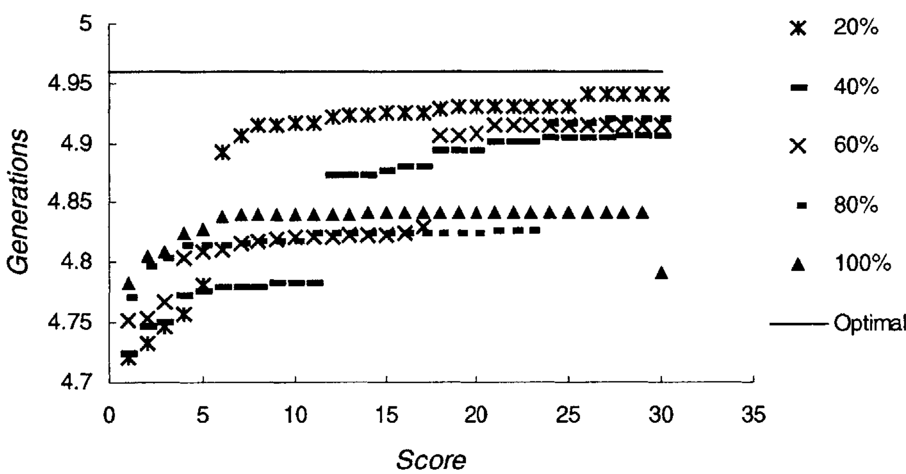


Figure 4.8 Plot of scores for GA/ANN runs at varied changeover points for a Himmelbau ANN of MSE 0.002

Figs. 4.7 and 4.8 show the best-of-generation scores for every generation for 5 selected changeover points. The final score for 100% changeover drops significantly in Fig. 4.8 when compared with Fig. 4.7. This behavior increases even more significantly with nets *less accurate than that in Fig. 4.8 as can be seen in Fig. 4.6. The net with an accuracy of 0.002 had only 4 out of the 10 runs within the 1% region: at 10%, 20%, 30% and 80% (Fig. 4.8). The other points had final scores deviating up to 4.79 (-3.4%) for the GA/ANN with a changeover point of 100%.*

The results also showed that less accurate nets with early changeovers still had good final scores (MSE 0.003, 0.005). This was likely to be the effect of early convergence on promising peaks, but not over-convergence such that potentially miss-mapped peaks dominated the final population.

It was also found that the final optimal score of some of the inaccurate nets when used for the entire optimization (100% of 30 generations) produced results that were worse than a normal optimization consisting of only 3 generations that started off with a random population! This occurred also for inaccurate nets that were used partially (changeover of less than 100%). For example, a net with an accuracy of 0.003 had a score of 4.67 three generations after changeover. A normal GA optimization that started with a random population had a score of 4.86 after three generations. It showed that better results were obtained in less time by not using the ANN for the optimization.

From the overview, we see that the accuracy of the nets affects the final optimization score in different ways. Nets with higher accuracy produced final best-of-generation results that were close to that of a normal optimization for all changeover points. Nets of lower accuracy had good final best-of-generation results for early changeovers, but poor results when used for longer periods of time. Nets of poor accuracy produced results that were actually worse than starting off with a random population, even for early changeover points. We can classify these nets into 3 categories (Chapter 3.2).

4.2.2 Upper bound determination

Keeping in mind the 3 categories of ANNs, we seek to find the upper bound for both test functions in this section. This was achieved by comparing the final results of a normal optimization to that of nets of varying accuracy. A least-squares data fit (Appendix A) of MSE and the percentage deviation from the optimal score produced the correlation of MSE and optimization accuracy. Given our upper bound definition to be not more than 1% away from the optimal, we obtain the MSE at 1%. The upper bound of a very accurate ANN for the Himmelbau was an MSE of 0.00056. The upper bound for the Hilly function was undetermined due to the difficulty of training the Hilly net beyond a MSE of 0.003 (Fig. 4.9). The results with this net deviated by 7% from the optimal.

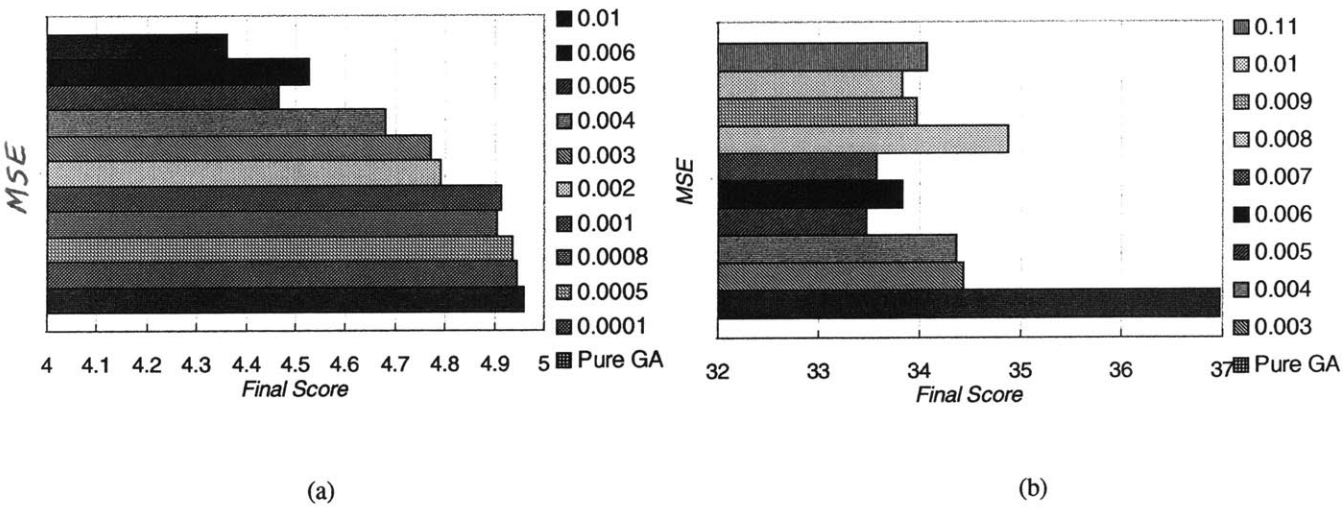


Figure 4.9 Bar chart of final best-of-generation scores (100% point) for (a) Himmelbau and (b) Hilly

At an MSE of 0.003 for the Hilly function, the optimal score was 34.4 compared to the normal GA score of 37.0, a deviation of 7% (Fig. 4.9).

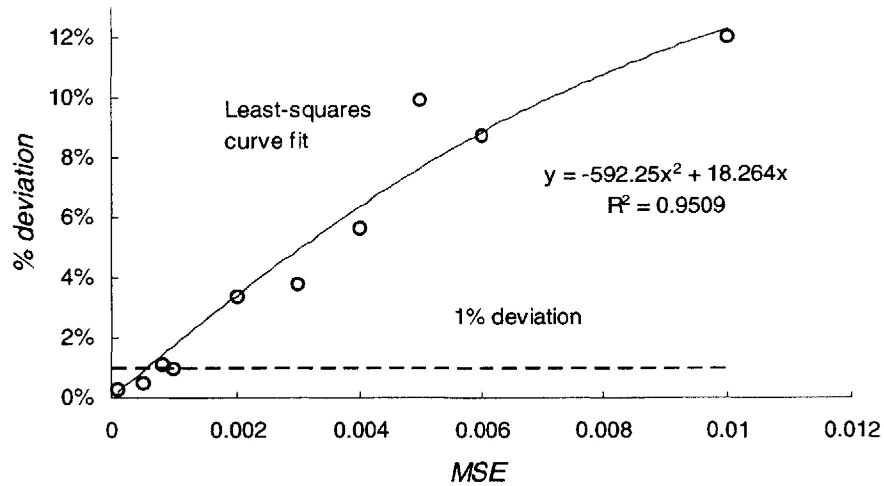
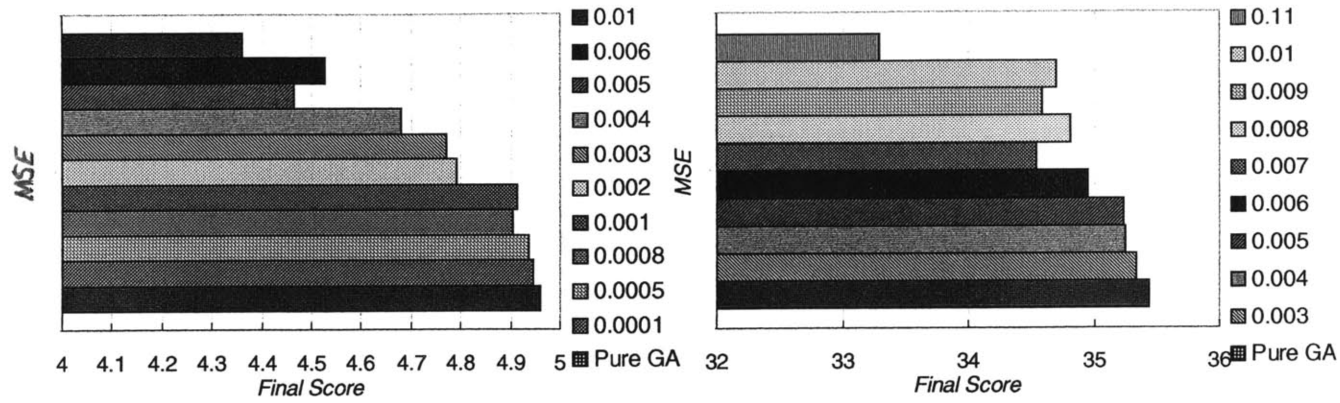


Fig. 4.10 Plot of percentage deviation from final optimal for the Himmelbau

A second order least-squares polynomial curve fit had its intersection with the upper bound deviation of 1% at an MSE of 0.00056 for the Himmelbau..

4.2.3 Lower bound determination

The lower bounds were similarly determined as the upper bound except for the reference normal score. In Chapter 3, we defined the lower bound as a net that produced an optimization at 10% (of the generations) that deviated less than 1% from the score of a normal optimization at the same number of generations. Using this measure, percentage deviations were obtained and curve-fitted against their MSEs to obtain the corresponding relation. The intersection of this curve with the 1% deviation point gave the lower bound. The lower bounds for the Himmelbau and Hilly function were at 0.0018 (Fig. 4.11) and 0.0057 (Fig. 4.12) respectively. It was noted that the lower bounds of the Hilly were significantly higher, implying the possibility that less accurate nets were needed for complex functions with peaks of large differentials than for simpler smoother functions. This also showed that there was no absolute MSE value that could be used to predict hybrid optimization success.



(a) (b)

Figure 4.11 Bar chart of best-of-generation scores at 10% changeover for (a) Himmelbau and (b) Hilly

As the nets become less accurate, the optimal score decreased The rate of decrease was less for the Hilly function (b) after an MSE of 0.003 than for the Himmelbau..

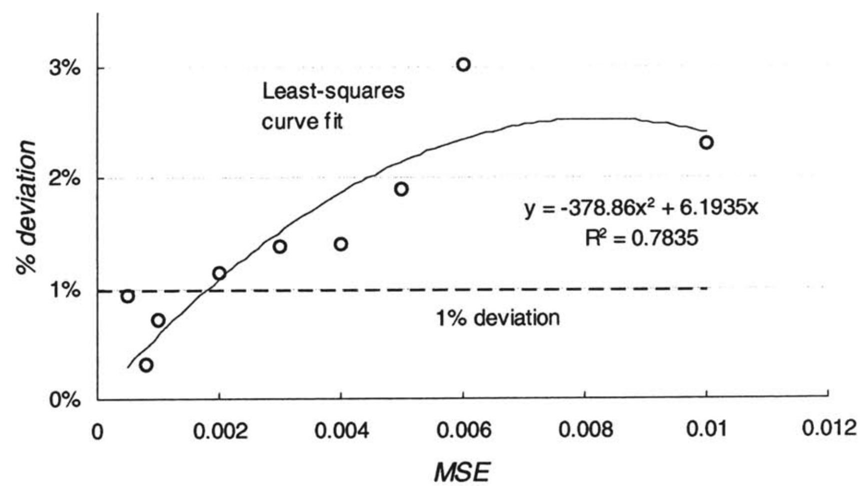


Fig. 4.12 Plot of percentage deviation from the 10% optimal score for the Himmelbau

A second order least-squares polynomial curve fit had its intersection with the upper bound deviation of 1% at an MSE of 0.0018 for the Himmelbau.

The lower bound MSE for the Hilly function was 0.0057 (Fig. 4.13). This was much higher than the lower bound of 0.0018 for the Himmelbau ANN. This appeared counterintuitive given that the Hilly function was a more complex. The GA should have required a more, not less, accurate neural net to obtain comparably favorable results. The explanation for this can be found in inspecting the search space of both functions. For the Himmelbau (Fig. 4.1 and 4.4), slight mapping inaccuracies could have created new peaks that were equivalent in height to some of the lower maxima given the functions relatively smooth peak transitions and lower height differential. This is contrasted to the Hilly function (Fig. 4.2 and 4.5) – the peaks here are in discrete locations with steep gradients and sharp height differentials. For this function, large absolute inaccuracies could occur in mapping the exact height of these maxima, but these maxima still stood out clearly from the rest of the landscape. This would allow the GA to still identify these peaks as maxima as a ripple function applied throughout (for randomness) would have a significantly smaller effect on the Hilly function than on the Himmelbau.

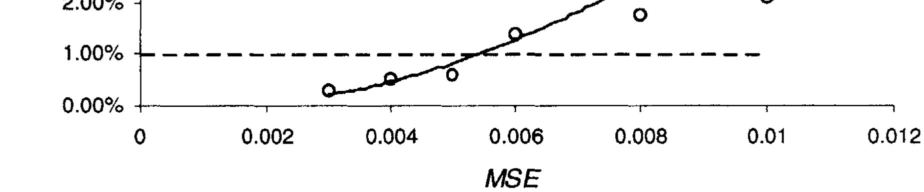


Fig. 4.13 Plot of percentage deviation from the 10% optimal score for the Hilly

A second order least-squares polynomial curve fit had its intersection with the upper bound deviation of 1% at an MSE of 0.0057 for the Hilly.

4.2.4 Changeover region

The **changeover region** was defined in chapter 3 as being the region where nets could be partially used to quickly obtain a promising population before switching over at some point in time during the optimization to use the actual function evaluation. This would enable time savings during the initial optimization phase but yet prevent the population from being overly converged on potentially false peaks. An equation was obtained that computed the changeover point of the net given its mean-squared error. This equation was tested against a map of data points, and the results showed that 83% of the scores fell within the predicted regions of accuracy (1% deviation).

The changeover region was obtained by simply doing a linear plot between the upper and lower bound. Fig. 4.14 shows the linear plot from the upper bound of 0.0018 to 0.00056. The linear equation predicts that any combination of MSE and changeover points along it will have a changeover score within 1% deviation from a normal optimization score at that same generation. The line also demarcates the various MSE/changeover combinations into 2 distinct categories. Any combination below the line means that the changeover score would be within 1% or less of the normal score, whereas any combination above the line would have a score with a deviating 1% and more.

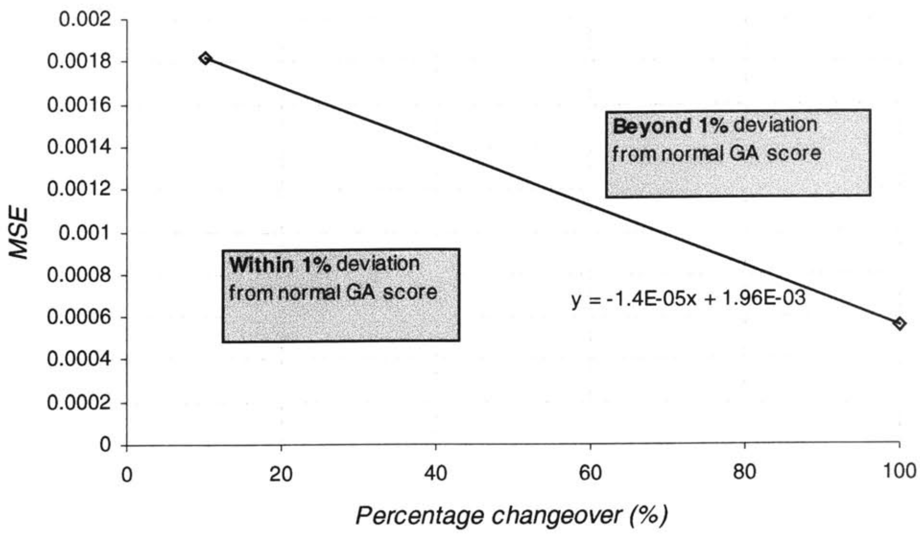


Fig. 4.14 Mapping of MSE to percentage changeover point for Himmelbau

From the linear equation in Fig. 4.14, we compute the changeover point with the changeover equation :

$$\%changeover = \frac{(MSE - 1.96x10^{-3})}{-1.4x10^{-5}} \qquad \text{Eq. (4.3)}$$

To test the accuracy of the changeover equation and the zones of demarcation, a test that included GA/ANN optimization data that was not used before was conducted. Fig. 4.15 shows the results of this test. The dark data points indicate that the deviations of the score were within the expected predictions and the hollow data points indicate that the scores were not within the expected predictions. The results show 87% of the data points falling into the proper demarcation zones. Of the 8 points that did not, the mean difference (from 1%) was 0.19% with a standard deviation of 0.21.

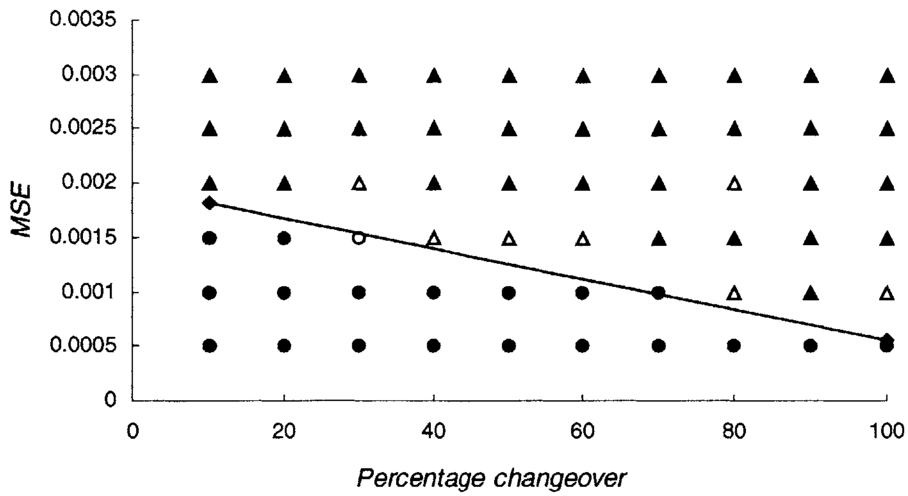


Fig. 4.15 Map of data points showing actual deviations of MSE/changover combinations.

Triangular data points are the regions in which the changeover score deviations were above 1% and round points are in the regions

The changeover equation computed the percentage changeover point given a neural net's accuracy (MSE). This computed changeover point predicted that optimization scores after changeover would be within the specified range of deviation (1%) from a normal optimization of the same number of generations. 87% of the data points tested against this equation produced results as predicted.

4.3 GA population statistics

In looking at the statistics of standard deviation and diversity of the population, we wanted to investigate the possibility of them being an indication of convergence of the population. Such an indication would enable us to use these measures (either in whole or in part) as a decision variable to turn off the neural net after a certain threshold had been reached, indicating a certain level of convergence, but not yet overly-converged.

Potential indications were the absolute value of standard deviation, absolute gradient of the deviation plots, rate of gradient decrease and diversity values.

4.3.1 Standard deviation

From selected results of standard deviation for the Himmelbau and Hilly are shown in Fig. 4.16 and 4.17 respectively, there were no visible measurements that provided us guidelines on when to changeover, though there were certain observed trends that will be discussed.

There was no absolute standard deviation value that could be generalized as an indicator, though the Hilly function exhibited some potential. Standard deviation values tend to be closely related to the landscape of each function. For example, though the standard deviation for both the Hilly and Himmelbau population tended to decrease as the optimization progressed, the population standard deviation for Shekel's foxholes optimization increased as the optimization progressed (not shown here). This occurred as a large part of the function landscape for Shekel's foxhole's was flat, giving an initially small standard deviation. But as the population started exploiting peaks that were found, the scores became increasingly divergent. This already argued strongly against the use of standard deviation as a measure of convergence.

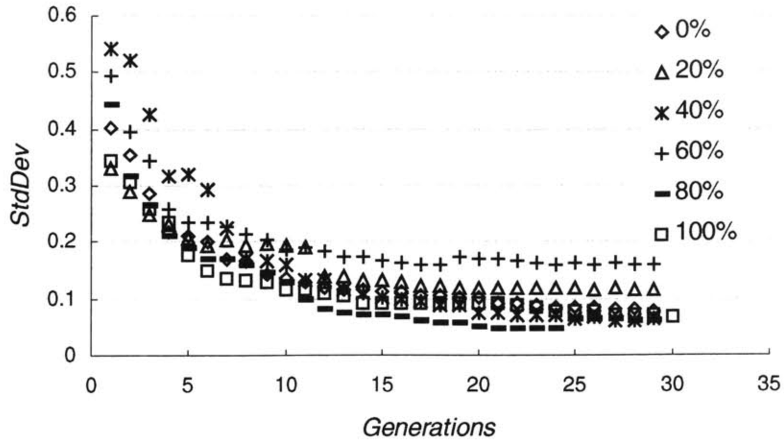


Fig. 4.16 (a) Plot of standard deviation of population for Himmelbau GA/ANN of MSE 0.0005

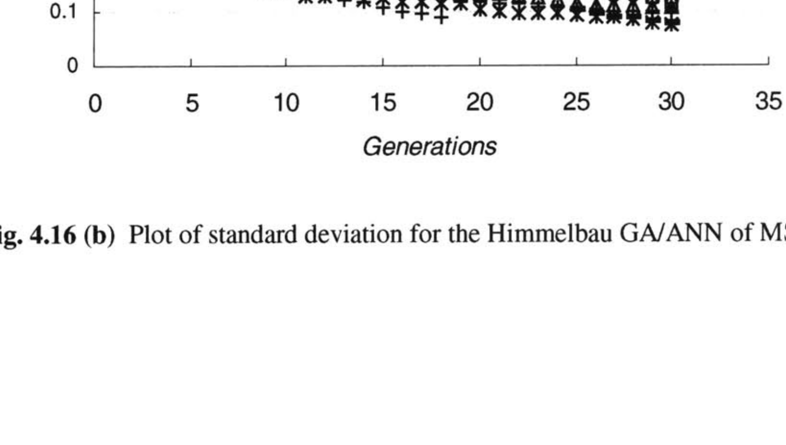


Fig. 4.16 (b) Plot of standard deviation for the Himmelbau GA/ANN of MSE 0.001

For the Hilly function, the standard deviation for the normal optimization was significantly higher than that of a hybrid optimization. While we know that a generalized measure of standard deviation cannot be applied as a changeover indicator as mentioned earlier, Figs. 4.17 (a) and (b) show a normal optimization maintaining a rather high standard deviation as compared to a hybrid optimization. The hybrid optimizations had standard deviations that were significantly lower in all cases (different changeover points). A search was made to relate the changeover scores to the standard deviation measures. Each single changeover score was examined and compared against the normal score. Most results showed significant decrease in the changeover score , but several counterexamples were found. For example, in Fig. 4.17 (a), the 40% the changeover score was 36.7 at the 60th generation, while the corresponding normal optimization score was 36.0. This showed that even at a much lower initial standard deviation, the changeover score could still be higher than that of a normal optimization (with higher standard deviations).

It is likely that for a normal optimization, the presence of a higher number and more accurate peaks enabled individuals to position themselves in a more distributed fashion than for a hybrid optimization.

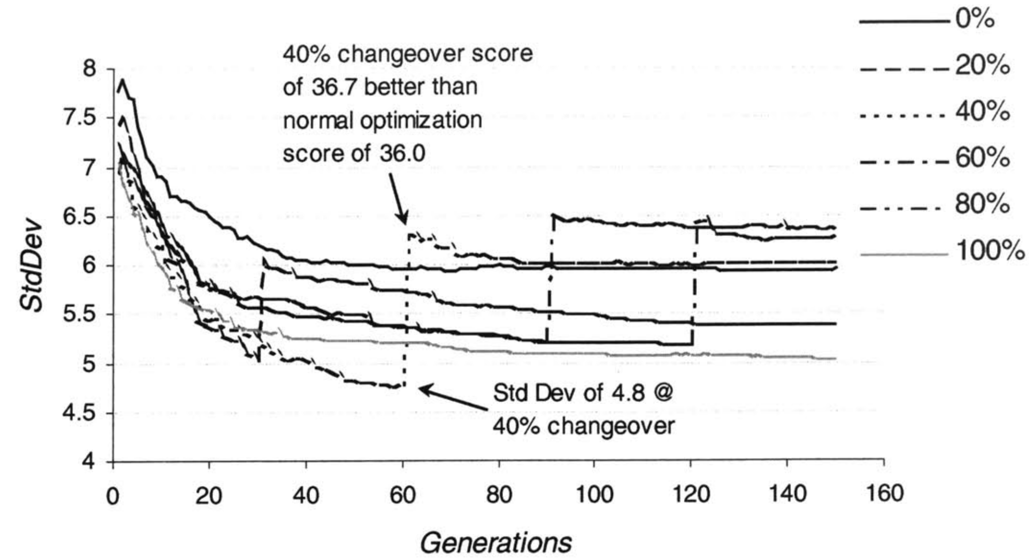


Fig. 4.17 (a) Plot of standard deviation for the Hilly GA/ANN of MSE 0.003

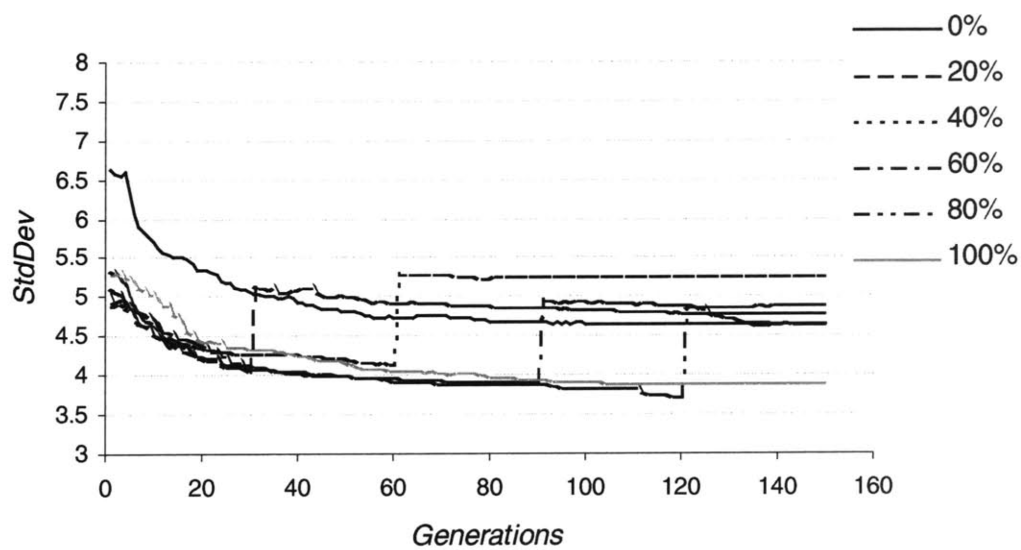


Fig. 4.17 (b) Plot of standard deviation for the Hilly GA/ANN of MSE 0.007

From the above results, it is clear that standard deviation does not provide clues to the accuracy of the hybrid optimization and nor does it provide indicators for a changeover point.

4.3.2 Diversity

The measure of diversity was another possible changeover indicator. Theoretically, this is rather remote as functions with multiple peaks can have a larger diversity and still maintain a healthy population while a function with fewer peaks can have a small diversity and still be considered rather optimal. However, possible relationships with the rate of diversity decrease were still worth investigating. Fig. 4.18 (a) and (b) show the diversity plots for a Himmelbau function with different accuracy.

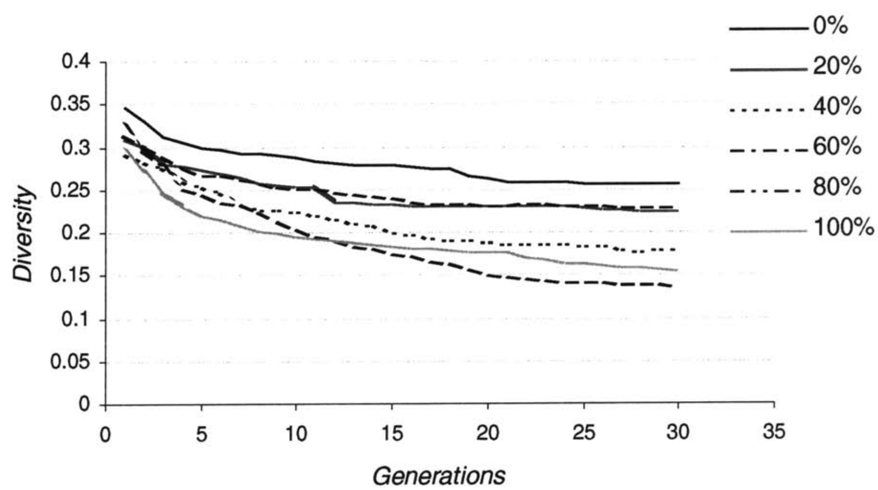


Fig. 4.18 (a) Plot of diversity for the Himmelbau ANN with MSE of 0.0005

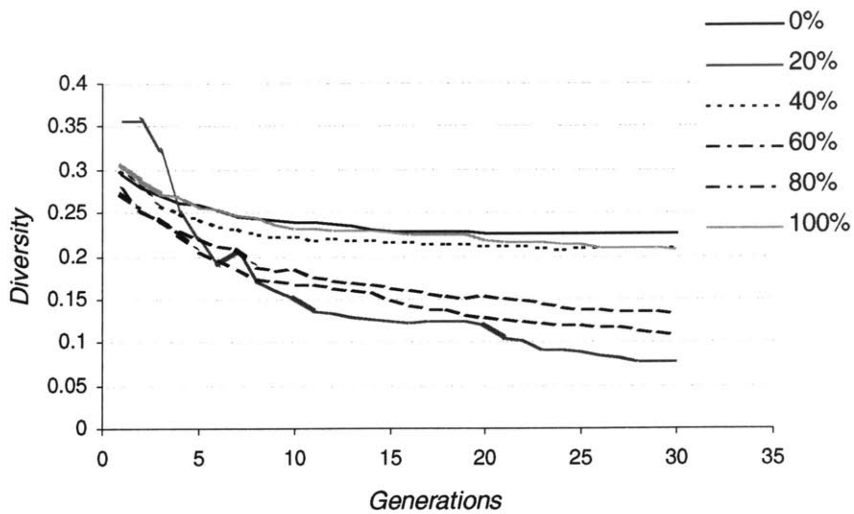


Fig. 4.18 (b) Plot of diversity for the Himmelbau ANN with MSE of 0.001

As expected, diversity statistics yielded no visible trend in either absolute value or rate. Fig. 4.18 shows the diversity measurements for various ANNs.

4.4 GA parameters

Given the very small computation time for an ANN output, doubling the population during the ANN phase provided a superior population to choose from during changeover while keeping computation times low. The final scores for a hybrid optimization in the population was double during the ANN optimization phase had final best-of-generation scores that were 12% higher than those with a normal population size.

Fig. 4.19 shows the results of the hybrid optimization experiment using the Hilly function with a net of 0.005 MSE. The first run was with a normal population size of 30. Runs with early changeover points of 5% and 10% were recorded. The second run was with a population size of 60 during the ANN optimization phase. After changeover, the GA recomputed the population scores and selected the top 50% of the individuals to go into the post-changeover generation (reduced population size of 30).

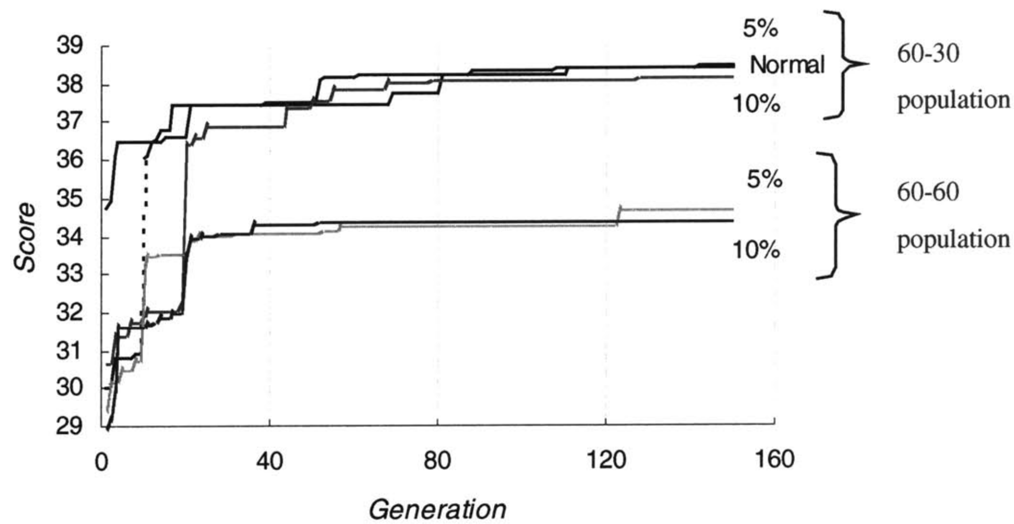


Fig. 4.19 Effect of increased population(2x) on performance of hybrid optimization for Hilly function

5. Conclusion

Using neural nets as meta-models for a genetic algorithm optimization produced very good results when accurate neural nets were considered, but produce extremely poor results with inaccurate nets that were worse than results obtained even after only a few generations of a normal GA optimization. In such a hybrid optimization, a decision is made to use each net fully, partially or not at all depending upon its accuracy. The optimization starts off with an increased population size that is reduced once the first net has been turned off. Each net is then turned off independently at a heuristically pre-computed time during the optimization depending on upon its accuracy.

Complex functions with sharp peaks were also shown to require a less accurate ANN in obtaining results of desired optimality than simple smooth functions. The four factors of ANN accuracy, convergence, service time and sensitivity were discussed and a framework was presented to analyze the effectiveness of artificial neural networks for use in genetic optimization. Given the accuracy of the neural network, it was possible to decide whether to use the net, not to use it or to compute the length of time for usage in an optimization. The framework effectively predicted the accuracy of the optimization results of the hybrid GA/ANN optimization when the computation heuristic was used 87% of the time. The framework was also shown to work with multiple ANNs for the more accurate network combinations. There was no absolute net accuracy value that could be used to guarantee good results, and the accuracy of the net was dependent on the complexity of the function. It appeared that more complex functions with peaks of large differentials required a less accurate neural net to obtain good hybrid optimizations as a less complex function. Population parameters like standard deviation and diversity provided no indications that allowed decisions to be made about when to use or stop using a net. Doubling the population during the optimization phase that used the neural nets provided a superior set of results 12% higher at a low computational cost. Dominance and time creep when using multiple neural nets can also further optimize time savings when using multiple nets in a large scale optimization with multiple variables.

Limitations and future work

The framework provided, while working well for our test function, requires the computation of the boundaries for each ANN. The changeover point computations were also based upon a percentage run. It is unclear if the boundaries are valid if the number of generations are significantly increased (e.g. 10% changeover for a 100-generation run is equivalent to a 100% for a 10 generation run, both theoretically giving the same results). Another key problem would be the estimation of boundaries and the final optimal value before actual optimization, both of which are necessary to compute the boundaries and the heuristic equation. Sensitivity measures would also enhance the usage of ANNs by allowing longer runs on ANNs which the objective function is insensitive to.

Appendix A

The Least Squares Approach of data fitting uses regression and the concept of minimizing the square of the errors between the least-squares curve and each individual data point. The accuracy of this approach is given by the **coefficient of determination** (R^2). This measurement represents the proportion of the data point deviations that can be attributed to the least-squares curve, so a R^2 value of 0.40 means that 40% of the sum of square deviations of the data about their mean can be attributed to the least squares curve-fit. A R^2 value of 1 means that the curve fit is perfect and a value of zero means that the curve fit has no correlation. The mathematical notation is given below :

First the sum of squares error of the data points about their mean is obtained :

$$SS_{yy} = \sum (yi - \bar{y})^2$$

Using data fitting, we obtain the curve and also the sum of squares error of the data points about the curve-fit :

$$SSE = \sum (y - \hat{y}_i)^2$$

The coefficient of correlation relates the difference in the ratios of error about the mean and about the curve fit according to the following equation :

$$R^2 = \frac{SS_{yy} - SSE}{SS_{yy}} = 1 - \frac{SSE}{SS_{yy}}$$

References

Beasley, David, 1993, “An overview of genetic algorithms : Part 1, Fundamentals”, Department of Computing Mathematics, Unversity of Cardiff.

Beasley, David, 1993, “An overview of genetic algorithms : Part 2, Research Topics”, Department of Computing Mathematics, Unversity of Cardiff.

Davis, Lawrence, (Editor) 1987, *Genetic Algorithms and Simulated Annealing,* Morgan Kaufman Publishers, California.

Deniz, Juan, 1998, “Application of Surrogate Neural Networks to Reduce the Computation time of Genetic Algorithms”, MIT CADLAB internal paper.

El-Beltagy M. A., Nair, P. B. and Keane A. J., 1999, ``Metamodeling Techniques for Evolutionary Optimization of Computationally Expensive Problems : Promises and Limitations," Proceedings of the Eighth International Conference on Genetic Algorithms, Orlando, FL, July, 1999.

Goldberg, David, 1989, *Genetic Algorithms in search, optimization and machine learning*, Addison Wesley.

Ignizio, James P., 1982, *Linear Programming in single- and multiple-objective systems,* Prentice Hall, New Jersey.

Lewis, Robert Michael, 1997, “A nonlinear programming perspective on sensitivity calculations for systems governed by state equations”, NASA Langeley Research Center, Hampton, VA. http://techreports.larc.nasa.gov/ltrs/Output/math.html

Lin Shyh-Chang, Punch III W.F., Goodman E.D., 1995, “A hybrid model utilizing genetic algorithms and Hopfield Neural Networks for Function Optimization”, Genetic Algorithm Research and Applications Group, Michigan State University.

Lorenzen Thomas, Anderson, V., 1993, *Design of Experiments – A No-Name Approach,* Marcel Dekker, Inc. New York.

Masters, T., 1993, *Practical Neural Networks Recipes in C++*, Academic Press.
Montgomery, D.C., 1997, *Design and Analysis of Experiments,* Fourth Edition, John Wiley & Sons, New York.

Pipe, A.G., Jin Y., Winfield A., 1994 “A hybrid adaptive heuristic critic architecture for learning in large static search spaces”, *1994 IEEE International Symposium on Intelligent Control,* Columbus, Ohio, USA, 16-18 August, 1994.

Ratle, A., 1998, "Accelerating the convergence of Evolutionary Algorithms by Fitness Landscape Approximation", Parallel Probem Solving from Solving from Nature-PPSN V.

Rogers, Leah L., 1992, "Optimal groundwater remediation using artificial neural networks and the genetic algorithm", Ph.D. thesis submitted to the Department of Applied Earth Sciences, Stanford University.

Schalkoff, Robert, 1997, *Artificial Neural Networks*, McGraw-Hill.

Schmidt, Stephen R., Launsby, Robert G., 1992, *Understanding industrial designed experiments,* 3rd Edition, Air Academy Press, Colorado Springs, Colorado.

Senin, N., Wallace D.R. and Jakiela, M., 1997, "A framework for mixed parametric and catalog-based design problem modeling and optimization", MIT CADLAB internal report.

Senin, N., Wallace D.R. and Borland, N., 1999, "Object-based design modelling and optimization with genetic algorithms", GECCO99 – Real World Applications.

Simpson, T., Peplinski J., Koch P., Allen J., 1997, "On the use of statistics in design and the implications for deterministic computer experiments", *1997 ASME Design Engineering Technical Conferences,* Sacramento, California, USA, 14-17 September.

Simpson, T. W., 1998, "Comparison of Response Surface and Kirging Models in the Muldidisciplinary Design of an Aerospike Nozzle". ICASE Report No. 98-16, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center.

Sousa, Ines, Wallace D., Borland N., Deniz J., 1999, *"A learning surrogate LCA model for integrated product design"*, MIT CADLAB internal report.

Wallace, David, Abrahamson, S., Senin, N., Sferro, P., 1999, "Integrated Design in a Service Marketplace", MIT CADLAB internal report.