CRANFIELD UNIVERSITY

MIKE J. W. RILEY

# EVALUATING CASCADE CORRELATION NEURAL NETWORKS FOR SURROGATE MODELLING NEEDS AND ENHANCING THE NIMROD/O TOOLKIT FOR MULTI-OBJECTIVE OPTIMISATION

SCHOOL OF ENGINEERING

PhD Thesis
Academic Year: 2010 - 2011

Supervisor: Dr. Karl Jenkins
March 2011

CRANFIELD UNIVERSITY

SCHOOL OF ENGINEERING

PhD Thesis

Academic Year 2010 - 2011

MIKE J. W. RILEY

# Evaluating Cascade Correlation neural networks for surrogate modelling needs and enhancing the Nimrod/O toolkit for multi-objective optimisation

Supervisor: Dr. Karl Jenkins

March 2011

# ABSTRACT

Engineering design often requires the optimisation of multiple objectives, and becomes significantly more difficult and time consuming when the response surfaces are multimodal, rather than unimodal. A surrogate model, also known as a metamodel, can be used to replace expensive computer simulations, accelerating single and multi-objective optimisation and the exploration of new design concepts. The main research focus of this work is to investigate the use of a neural network surrogate model to improve optimisation of multimodal surfaces.

Several significant contributions derive from evaluating the Cascade Correlation neural network as the basis of a surrogate model. The contributions to the neural network community ultimately outnumber those to the optimisation community.

The effects of training this surrogate on multimodal test functions are explored. The Cascade Correlation neural network is shown to map poorly such response surfaces. A hypothesis for this weakness is formulated and tested. A new subdivision technique is created that addresses this problem; however, this new technique requires excessively large datasets upon which to train.

The primary conclusion of this work is that Cascade Correlation neural networks form an unreliable basis for a surrogate model, despite successes reported in the literature.

A further contribution of this work is the enhancement of an open source optimisation toolkit, achieved by the first integration of a truly multi-objective optimisation algorithm.

Keywords: early stopping, ensembling, multimodal functions, variance, bias, subdivision technique, shape optimisation

# ACKNOWLEDGEMENTS

Working as a research student for the last three-plus years has been a largely pleasurable and enlightening experience. I must first thank my tutor, Dr. Karl Jenkins. Without him making me aware of this EPSRC funded PhD in 2007, I would never even have started this journey.

I want to acknowledge Monash University for the use of their Nimrod software (Chapter 6). The Nimrod project has been funded by the Australian Research Council and a number of Australian Government agencies, and was initially developed by the Distributed Systems Technology CRC.

I want to thank the many academics and students who have given sometimes minutes, sometimes many hours of their help and advice to me – you know who you are! A special mention goes to a good friend, Dr. Rick Drury, whose mental rigour and a devotion to all things academic has been inspirational. I am very grateful to Dr. Patrick Verdin who created the CFD meshing and solving files that enabled the aerofoil optimisation studies of Chapter 5. I also want to say a big “thank you” to Lia Hedley and to my parents, MaryRose and Bill Riley, who have kept me sane despite being driven to distraction by my endless ramblings on technologies that are of no real interest to them. Lastly, for the small group of people who will actually read this entire thesis, I hope that I can succeed in making some of it interesting, informative or useful for you.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF EQUATIONS

# 1 INTRODUCTION

The central theme of this thesis is to explore, develop, or enhance methods of reducing the computational load of optimisation with a particular focus on multimodal functions. Chapters 3 and 4 evaluate Cascade Correlation neural networks as surrogates for accelerating optimisations. Chapter 5 reduces the load of aerofoil optimisation by determining appropriate search domains for low Reynolds numbers cases. Chapter 6 enhances an existing optimisation toolkit by interfacing a multi-objective optimisation algorithm along with enabling parallelism in that algorithm.

A properly trained surrogate model delivers a good approximation of the objective function that would be returned by a high-fidelity model but much faster. This speed-up is the advantage of surrogate modelling: the surrogate model described in this work returns evaluations of objective functions in less than 10ms (200 MFLOP) irrespective of the problem dimensions. Machine learning algorithms are often used for model-approximation and the surrogate of this work is based on the Cascade Correlation neural network.

The principal challenge when training a neural network is to reduce both its *bias* (under fitting) and its *variance* (over fitting). Reducing the variance of the Cascade Correlation neural network forms the theme of Chapter 3, whilst bias is treated in Chapter 4. The absolute values of variance errors are problem dependent, however, Chapter 3 contains a detailed study of two existing techniques that, for the test functions used, are found to reduce variance by a factor of three. The work in this chapter produced the following contributions:

- Determining an appropriate number of training samples per dimension (3.3.1)
- The postulate that we may dispense with creating testing datasets, and thereby save a significant amount of time (3.3.3)
- A novel technique for determining the variance and bias of a neural network ensemble (3.3.4)

The novel method for determining bias and variance prompts further analysis and a statistical treatment is given in the appendix.

As the focus of this thesis is multimodal problems, and the motivation is to develop a CasCor metamodel for integration with Nimrod/O, Chapter 4 describes research on the performance of Cascade Correlation when mapping multimodal response surfaces. To the author's knowledge, there are no examples in the literature that explicitly evaluate the performance of CasCor on low dimensional (2-5), highly multimodal, surfaces.

This neural network type is found to exhibit a particular weakness on these surfaces. Despite the reductions in mean squared error from the variance-reducing methods in Chapter 3, undesirably high testing errors remain. It is shown that this neural network exhibits the problem of possessing a high bias (severe under-fitting). A new subdivision technique named 'patchworking' is introduced to address the - not previously published - bias problem of this neural network type; patchworking delivers significantly improved fits to multimodal surfaces. The contributions of this chapter are the identification of the bias problem of Cascade Correlation neural networks and the introduction of the patchworking technique to overcome this problem.

In addition to training Cascade Correlation neural networks on mathematical test functions, real world case studies were sought from the publicly-available machine learning repository [1]. Whilst not representative of optimisation problems, the concrete compressive strength and the abalone age-predictions are examples (Chapter 3) that do illustrate successful applications of this surrogate. In Chapter 4, a very large census dataset is used to illustrate the benefits of the patchworking algorithm on real-world data.

The competitive manufacturing climate in the last two decades has highlighted inadequacies in the serial practice of design. This competitive environment requires organisations to design high-quality products faster, better, and cheaper than their competitors [2]. In civil, mechanical, aerospace, and

electronic engineering, computer aided engineering (CAE) software has assisted the designer in achieving these goals.

The uses of CAE can encompass simulation, validation, and the optimisation of designs. A designer starts with the idea of a new product and uses computer aided design (CAD) software to create a preliminary design. With the use of computer based modelling tools, the preliminary design can be analysed for functionality as the design is being created. By manipulating the geometry of a design, its performance can be improved. “Performance” in this case is the improvement of some metric(s) determined by the engineer a priori. These metrics are better known as *objective functions*.

In shape optimisation, the optimisation algorithms manipulate the parameters that specify the geometry of a design, and complex models, typically incorporating solids and fluids solvers, return the objective functions. However, the computational load of evaluating these time-expensive objective functions can inhibit, or even prohibit, optimisation.

Chapter 5 applies the multi-objective optimisation algorithm (detailed in Chapter 6) to reduce the search space, and therefore the computational load, of aerofoil shape optimisation. The reverse chronology of these chapters acknowledges the minor contribution of this case study. Chapter 5 also sets the scene for the generic process of shape optimisation; the overall aim being to expose the practicalities of such work, with the intent of revealing potential research gaps.

One of the driving motivations for the current work was that the outcomes should aim to be of practical, as well as theoretical (3.3.3 , 3.3.4), use to the research community. For this reason an early decision was made to enhance an existent software toolkit rather than attempt to build another stand-alone package. The maturity of the Nimrod toolkit (Figure 1-1), its ease of use, a good working relationship with the developers, and access to its source code motivated the choice of this toolkit for this work.

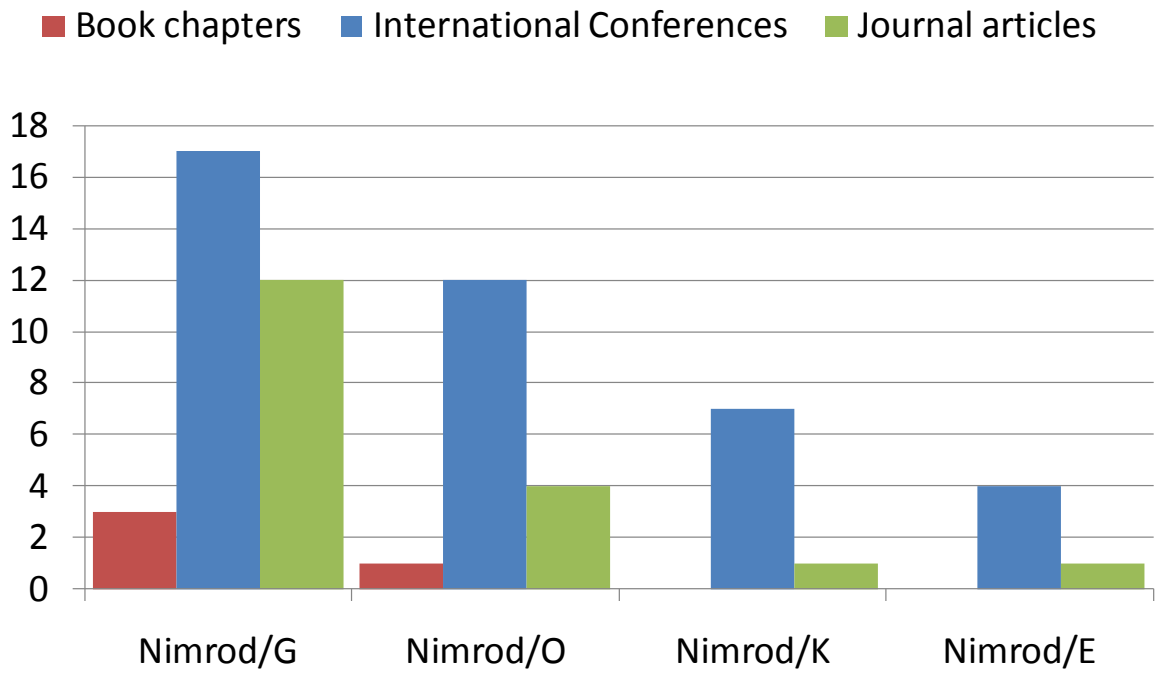



**Figure 1-1 Publications relating to the Nimrod toolkit**

Chapter 6 contains a significant contribution of this thesis. The Nimrod/O [3] optimisation package is part of a suite of problem solving tools developed since 1995 at Monash University, Melbourne, Australia. To date, this toolkit contains software packages for; Design of Experiments (Nimrod/E), Workflow management (Nimrod K), Grid Computing (Nimrod G), a web portal for job management (Nimrod/P), and an optimisation package, Nimrod/O. The work described in Chapter 6 details how a truly multi-objective optimisation algorithm was interfaced to Nimrod/O for the first time. Another contribution is the introduction of a parameter that will reduce dramatically the wall-clock time for these optimisations by enabling concurrent function evaluations. The successful implementation is illustrated with another shape optimisation; that of a rib-reinforced wall bracket.

The following papers have been published as a result of the research described in this thesis:

Journal paper

M. J. W. Riley, C. P. Thompson, and K. W. Jenkins, “A Study of Early Stopping, Ensembling, and Patchworking for Cascade Correlation Neural Networks”, IAENG International Journal of Applied Mathematics 40:4, 2010, pp. 307-316.

Conference papers

Riley, M. J. W., Peachey, T., Abramson D., and Jenkins, K. W., 2010, "Multi-objective engineering shape optimization using differential evolution interfaced to the Nimrod/O tool", IOP Conference Series: Materials Science and Engineering, Volume 10, Article Number 012189.

M. J. W. Riley, K. W. Jenkins, and C. P. Thompson, “Improving the Performance of Cascade Correlation Neural Networks on Multimodal Functions,” Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2010, WCE 2010, 30 June - 2 July, 2010, London, U.K., pp. 1980-1986. (Best paper award 2010 ICCSDE)

# 2 BACKGROUND

## 2.1 Meta/surrogate modelling

Metamodels, previously known as surrogate evaluation models (or just evaluation models) are currently active research areas in the optimisation of complex designs. Complex in this sense would mean those designs for which a single objective function evaluation is very (time) costly and, in many cases, these are designs that involve a high number of parameters (10+).
Metamodelling can play several different roles for the engineer [4] (Table 2-1).

**Table 2-1 The uses of metamodels**

| | |
|---|---|
| Model approximation | Approximation of computation-intensive processes across the entire design space, or global approximation, is used to reduce computational costs. |
| Design space exploration | The design space is explored to enhance the engineers' understanding of the design problem by working on a cheap-to-run metamodel. |
| Problem formulation | Based on an enhanced understanding of a design optimisation problem, the number and search range of design variables may be reduced; certain ineffective constraints may be removed; a single objective optimisation problem may be changed to a multi-objective optimisation problem or vice versa. Metamodelling can assist the formulation of an optimisation problem that is easier to solve or more accurate than otherwise. |
| Optimisation support | Industry has various optimisation needs, e.g., global optimisation, multi-objective optimisation, multidisciplinary design optimisation, probabilistic optimisation, and so on. Each type of optimisation has its own challenges. Metamodelling can be applied and integrated to solve various types of optimisation problems that involve computation-intensive functions. |

Once trained, surrogate models can replace expensive fluids or solids evaluation codes and facilitate multi-objective optimisation and the exploration

of new design concepts; returning objective function evaluations in fractions of a second.

Metamodelling involves:

1. Choosing an experimental design for generating the data (2.2)
2. Choosing a model to represent the data (2.3)
3. Fitting the model to the observed data from the experiments (2.6).

The metamodel of this work uses a neural network to represent the data. The data itself comes from sets of objective function evaluations, or ‘experiments’. The design of these experiments conforms to orthogonal sampling. The benefits of orthogonal sampling are discussed in section 2.2.3

## 2.2 Design of experiments

With the exclusion of trivial problems, for which full parameter sweeps can be performed, techniques from Design of Experiments (DoE) are typically applied for sampling a problem’s response surface. Three sampling techniques are outlined below; random sampling, Latin hypercube sampling and orthogonal array sampling.

### 2.2.1 Random sampling

The Direct Monte Carlo Sampling method, which is a random sampling method, is still popular in industry, regardless of its inefficiency. This popularity probably derives from the fact that the adequate and yet efficient sample size at the outset of metamodelling is unknown for any black box function. Therefore it holds an advantage over orthogonal and Latin hypercube sampling in that no decision as to the size of the sample is necessary at the outset [4]. The inefficiency of the technique derives from the fact that Direct Monte Carlo sampling has no ‘memory’ of previous samples.

For example, if we have a two dimensional problem to sample (variables X1 and X2) in the domain [0.0,1.0], and we generate 15 samples, it is possible to

find clustering of some of those samples in the input space (Figure 2-1) [5]. Clustered samples do not provide new information or insight into the overall behaviour of the response surface – moreover, the corollary is that clustering in one region leads to an undesirable sparseness of sampling in other regions of the domain.

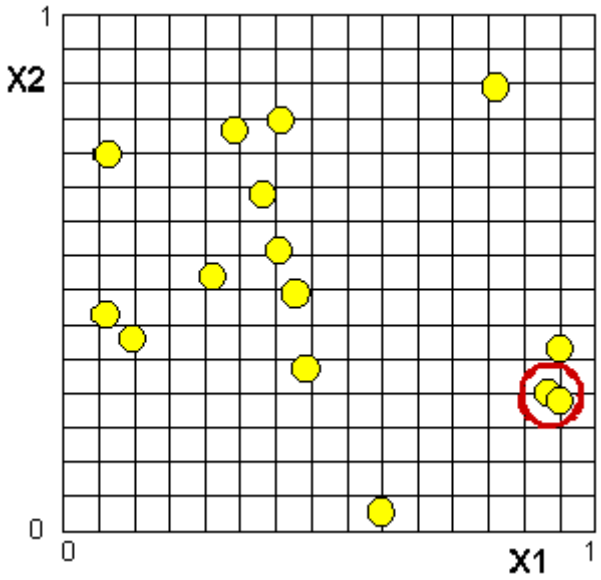



**Figure 2-1 Monte Carlo Sampling showing clustering (circled)**

### 2.2.2 Latin hypercube sampling

If we consider the sampling of a two dimensional function in the form of a grid of points, Latin hypercube sampling would consist of samples within that grid with each sample point existing at a unique x and a unique y co-ordinate. If the leading diagonal was populated with sampling points then we would have a Latin hypercube Design of Experiment – however, such a DoE would be undesirable as it would not be classed as space-filling. Figure 2-2 [5] shows an example of a space-filling Latin hypercube. Unlike Monte Carlo sampling, space-filling Latin hypercubes can be thought of as having a 'sample memory', meaning that it avoids repeating samples that have been evaluated before (i.e. avoiding clustering). Although dependent on the problem at hand, a space-filling Latin hypercube DoE could require 20% to 40% fewer samples than a Monte Carlo DoE to deliver the same results with the same accuracy [5].

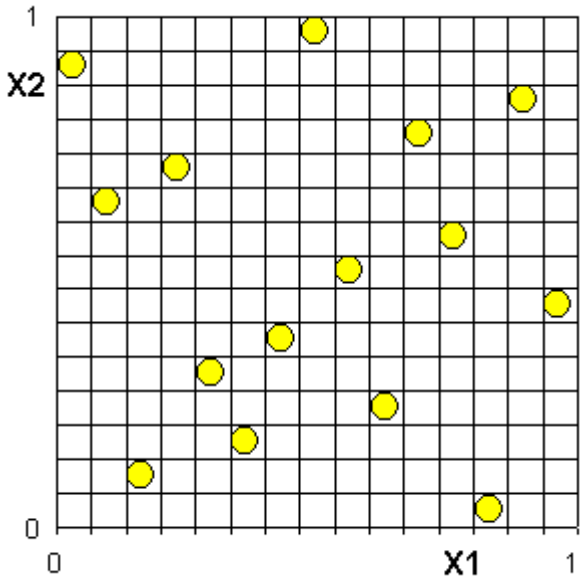



**Figure 2-2 Space-filling Latin hypercube**

### 2.2.3 Orthogonal Sampling

Orthogonal array testing is a systematic, statistical way of testing. The permutations of factor levels comprising a single treatment are chosen such that their responses are uncorrelated, each treatment thereby giving a unique piece of information. By creating a design of experiments based on an orthogonal array, that same piece of information is gathered in the minimum number of experiments.

Each orthogonal vector conveys different information from any other vector in the DoE, hence avoiding redundancy. Additionally, each of the vectors is statistically independent of the others, i.e. the correlation between them is nil.

Sampling with orthogonal arrays (OAs) can be described as a generalisation of Latin hypercube sampling whose one dimensional projection is uniformly spaced [6]. Wang [4] highlights the two most important properties of the sampling distribution of a DoE. Those are its orthogonality, and its space-filling properties. OAs enhance the ability to analyse and estimate as many effects and interactions as possible. Research into orthogonal array generation is an

ongoing subject in mathematics, though recent progress has yielded powerful algorithms [7].

An OA is defined in the form $OA.N.k.s.t$ indicating an orthogonal array with $N$ runs, $k$ factors, $s$ levels, and strength $t$. This is an array of size $N$ by $k$, with entries from 0 to $s-1$ with the property that in any of the $k$ columns each of the $s$ possibilities occurs equally often [8].

**Table 2-2 The orthogonal array used for this work (OA.25.6.5.2)**

| | 5 levels can be tested in up to 6 dimensions | | | | | |
|---|---|---|---|---|---|---|
| Experiment | **1** | **2** | **3** | **4** | **5** | **6** |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 1 | 2 | 3 | 4 |
| 3 | 0 | 2 | 2 | 3 | 4 | 1 |
| 4 | 0 | 3 | 3 | 4 | 1 | 2 |
| 5 | 0 | 4 | 4 | 1 | 2 | 3 |
| 6 | 1 | 0 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 2 | 4 | 0 | 3 |
| 8 | 1 | 2 | 4 | 0 | 3 | 2 |
| 9 | 1 | 3 | 0 | 3 | 2 | 4 |
| 10 | 1 | 4 | 3 | 2 | 4 | 0 |
| 11 | 2 | 0 | 2 | 2 | 2 | 2 |
| 12 | 2 | 1 | 4 | 3 | 1 | 0 |
| 13 | 2 | 2 | 3 | 1 | 0 | 4 |
| 14 | 2 | 3 | 1 | 0 | 4 | 3 |
| 15 | 2 | 4 | 0 | 4 | 3 | 1 |
| 16 | 3 | 0 | 3 | 3 | 3 | 3 |
| 17 | 3 | 1 | 0 | 1 | 4 | 2 |
| 18 | 3 | 2 | 1 | 4 | 2 | 0 |
| 19 | 3 | 3 | 4 | 2 | 0 | 1 |
| 20 | 3 | 4 | 2 | 0 | 1 | 4 |
| 21 | 4 | 0 | 4 | 4 | 4 | 4 |
| 22 | 4 | 1 | 3 | 0 | 2 | 1 |
| 23 | 4 | 2 | 0 | 2 | 1 | 3 |
| 24 | 4 | 3 | 2 | 1 | 3 | 0 |
| 25 | 4 | 4 | 1 | 3 | 0 | 2 |

### 2.2.4 Summary of design of experiments

For the sampling of the test functions in Chapters 3 and 4, orthogonal arrays were chosen to generate the training datasets. This method of sampling was

chosen as OAs can provide the convenient benefit of screening the number of dimensions of a problem (if necessary) with the use of ANOVA (ANalysis Of Variance).

## 2.3 Popular metamodel types

### 2.3.1 Response surface methodology (RSM)

RSM fits a response surface with some form of least squares linear regression (typically a low order polynomial is used although recent advances have seen Padé–Legendre approximations used successfully for discontinuous response surfaces [9]). RSM is a popular technique; partly due to the simplicity of its implementation, and partly because many examples exist in the literature. There are three main problems with RSM. Firstly, polynomial models cannot capture highly non-linear response variations; the accuracy of quadratic RSM being questionable for multimodal problems [10]. Secondly, if higher order polynomials are used, the large number of co-efficients to be determined results in large training times [11]. Thirdly, the amount of training data required to build a second order surface grows quadratically with the input dimensions of the problem. Hence, increasing the input parameters results in a rapid non-linear increase in the necessary training data [12].

### 2.3.2 Kriging

Kriging is also known as a Gaussian process or a Gaussian random function method [13]. Unlike the linear regression of RSM, it uses Bayesian regression. It has two significant advantages over RSM: 1. It can be applied for mapping surfaces for which there are a significant number of input parameters e.g. 20-30, and, 2. It can honour the training data; fitting it precisely by interpolation, or, smooth over the data, thus approximating the surface [14]. A weakness of the Kriging method is the general need to tune multiple hyperparameters that control curvature and the degree of regression; this can be very time consuming on large data sets in many dimensions [6].

### 2.3.3 Neural networks

Originally inspired by the multilayered information processing structures of biological brains, neural networks typically consist of a large number of simple, but interconnected, processing units. The processing units, called *neurons*, are multiple linear regression models with non-linear transformations applied to their inputs. The architecture of the network is formed by connecting many neurons with *weights* (the regression co-efficients). Hence, there are two main issues: 1. Specifying this architecture, and, 2. Training the neural network to perform well with respect to the training dataset [11].

The advantage of using neural networks is that they are universal functions approximators [15] i.e. the family of functions that the network can implement is broad enough to contain $f$ or a good approximation of $f$. For the training of a neural network to converge it must, in the limit, approach the target function as closely as desired. A sequence $\{f_n\}$ strongly converges to $f$ if $\lim_{n\to\infty} \| f - f_n \| = 0$, where $\| \cdot \|$ is the norm for the function space being considered [16].

Two criticisms often levelled are; that the training usually takes a significant amount of time, and, they are 'black-box' approximators; once trained, it is difficult to trace the behaviour, relationships and dynamics of the network back to the reference model [17].

### 2.3.4 Radial Basis Functions

Radial Basis Functions are closely related to both Kriging and neural networks. They approximate surfaces by using a linear combination of radially symmetric functions [18]. Like the Kriging method, they can exactly interpolate a surface from the training data. However, as all RBFs employ a measure of distance between data points, attempting to learn in a high dimensional space means that almost every sample is closer to the boundary of the domain than to another point. This makes Radial Basis Functions less suited to learning with a

very high number of input parameters [19]. As an example, take a 15 dimensional problem. The hypercube of the input space would have $2^{15}$ vertices (32,768) and $2^{(15-1)} \times 15 =$ 245,760 edges. A typical training dataset for such a problem would likely contain fewer than 245,760 samples; hence there would be many more edges of the domain than samples.

### 2.3.5 Support Vector and Relevance Vector machines for regression (SVR & RVM)

Both of these techniques are closely related and known as *sparse kernel methods*. They centre basis functions on subsets of the training data and then train on these subsets. A major advantage of these learning methods is that their mathematical formulation is dimension-independent. This makes them an attractive solution for learning in very high dimensional cases [17].

The advantage of SVR over RVM is that the training consists of the solution of a convex (i.e. simpler) optimisation problem [20]. Though SVR has been used successfully for surrogate modelling [18], it is disadvantaged by the necessity to determine two parameters after training. This post-training-optimisation is performed by a cross-validation method and is typically time consuming. A disadvantage of both types is that they map multivariate inputs to only a univariate output variable; hence further models must be trained if several objective functions are to be surrogated [21].

RVM has an identical functional form to SVR but, by reforming the support vector solution with ‘expectation maximisation learning’, the relevance vector machine is created. A Bayesian framework is used in the case of RVM, thus providing posterior probabilistic outputs, and typically much sparser solutions than for SVR; both of which are desirable as described in [20]. The advantage of the relevance vector method over the support vector is that there is no requirement to determine any parameters after training. However, this benefit is associated with a penalty – namely, that the training procedure now involves the solution of non-convex optimisation problem. When training an RVM (and neural networks), we face the ‘local minima problem’.

## 2.4 Local minima problem

The local minima problem [22] arises when attempting an optimisation on a function whose response surface is multimodal. There will be one or more local minima and there could be several global minima. The challenge posed to any optimisation algorithm is to find the location within the search space of a global minimum. This is a pertinent problem for the training algorithm of some surrogates and also for the optimisation of engineering designs. For example, radial basis functions and support vector regression are insensitive to the local minima problem [23]. However, relevance vector machines and neural networks are subject to this problem.

Distinct from the nature of the objective function to be mapped, it is the *error surface* of a neural network that will typically have multiple local minima. The total number of local minima is compounded by symmetries in the network. For example, taking the case of a network with two layers of weights, $M$ hidden units, and a sigmoid activation function, there will be a family of $M!2^M$ equivalent minima belonging to each distinct local minima [24].

In the training of a neural network, the difference between a neural network's output and the desired output is the error that should be minimised. It is by altering the weights of a neural network that this is achieved. For example, the backpropagation training algorithm approaches this problem by means of a gradient descent method but, as such, the training is subject to convergence to a local minimum – rather than the desired global minimum.

## 2.5 Cascade correlation neural network

The Cascade Correlation neural network (also referred to as CasCor in this work) is a constructive neural network. Growing on-demand, it only adds hidden neurons as and when they are needed. The standard CasCor network adds each new neuron to a new layer, creating deep neural networks. Neurons themselves can be thought of as *feature detectors*; the more features that exist in the response surface of a target function, the more neurons will be necessary

to map that surface. With insufficient numbers of neurons, too few features can be represented and the network will possess an undesirably high bias (Figure 2-3). Hand-crafting the topology of a neural network is a very time consuming process, and so constructive neural networks that solve this problem automatically have become very popular.

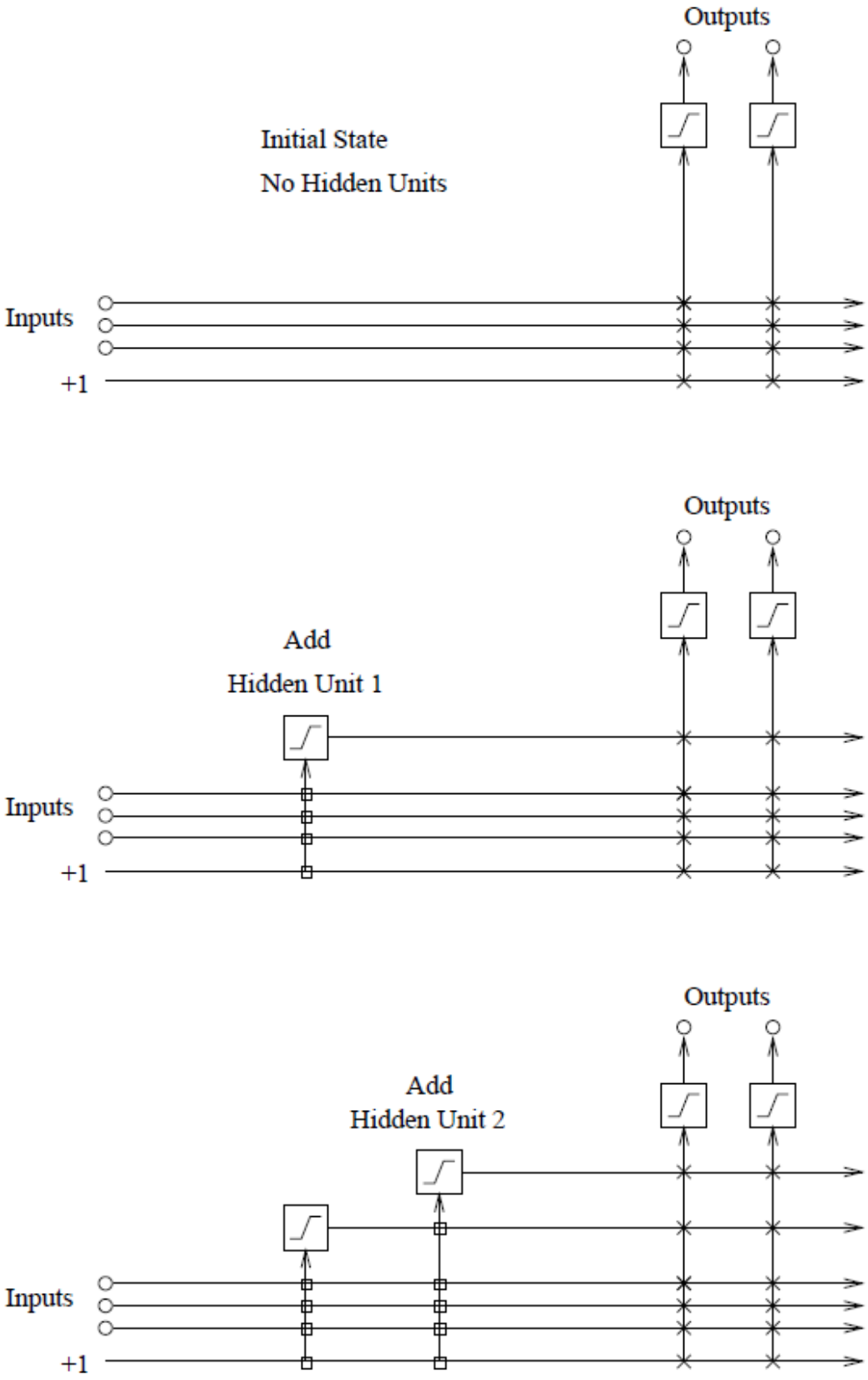



**Figure 2-3 Cascade Correlation training**

In Figure 2-3 [25] the Cascading architecture is displayed for a neural network with three input and two output dimensions. The diagrams show the initial state, and then the addition of two hidden units. The vertical lines sum all incoming activations. Boxed connections are frozen, X connections are trained repeatedly. The “+1” input is known as the “bias neuron”. During training, each neuron begins as a *candidate neuron*. It is not yet connected to the network. For the current work a pool of candidates is used, with each unit having a different activation function and different random initial weights. All receive the same input signals during training, but do not interact with each other. When the optimal candidate is inserted as the next hidden unit the other candidate neurons for that layer are discarded. The activation functions used in this work are as follows: Sigmoid, Sigmoid Symmetric, Gaussian, Gaussian Symmetric, Elliot, Elliot Symmetric. They share the necessary property of being differentiable.

Fahlman and Lebiere [25] describe this neural network as follows: training progresses by running a number of passes over the data in the training set. Each candidate input weights are adjusted after each pass to maximise $S$, the sum over the output units ($o$) of the magnitude of the correlation between $V$ ,the candidate unit’s value and $E_o$ , the residual output error observed at unit $o$. $S$ is defined as:

$$S = \sum_o \left| \sum_p \left( V_p - \bar{V} \right) \left( E_{p,o} - \bar{E}_o \right) \right|$$

Where $o$ is the network output at which the error is measured and $p$ is the training pattern. The quantities $\bar{V}$ and $\bar{E}_o$ are the values of $V$ and $E_o$ averaged over all patterns. In maximising $S$, $\partial S / \partial w_i$ must be computed i.e. the partial derivative of $S$ with respect to each of the candidate unit’s weights ($w_i$). Expanding and differentiating the formula for $S$ gives:

$\partial S/\partial w_i = \sum_{p,o} \sigma_o \left( E_{p,o} - \bar{E}_o \right) f_p^{'} I_{i,p}$ where $\sigma_o$ is the sign of the correlation between the candidate's value and output $o$, $f_p^{'}$ is the derivative for pattern $p$ of the candidate unit's activation function with respect to the sum of its inputs, and, $I_{i,p}$ is the input the candidate unit receives from unit $i$ for pattern $p$. After $\partial S/\partial w_i$ has been computed for each incoming connection, a gradient ascent is performed to maximise $S$. When $S$ stops improving, the best candidate is installed to the network and its input weights are frozen (*weight freezing*). More hidden neurons as installed with the above cycle above until one of the used-defined stopping criteria are met and the network is pronounced as "trained". Drago and Ridella [26] also investigated the convergence properties of the Cascade Correlation neural network and proved a speed of the order $O(1/n_h)$ where $n_h$ is the number of hidden neurons.

One cannot traverse far through the Cascade Correlation literature without meeting a reference to its notable performance on the "two-spirals problem": 397 articles are returned by a Google scholar search for '"cascade correlation" +spirals'. This is a problem that is said to pose a very difficult learning benchmark for backpropagation neural networks, but one for which Cascade Correlation performs very well [27]. Arguably, this notable performance has singularly popularised this neural network type more than any other benchmark. Seen in Figure 2-1 are the results of a count of the literary references to this neural network since 1990. In red are the number of articles published annually that have 'Cascade Correlation' in their title. The blue columns show that, each year since 1993, this neural network has been referred to over 100 times in the body of papers indexed on Google scholar, showing that this neural network is still active in the research community.

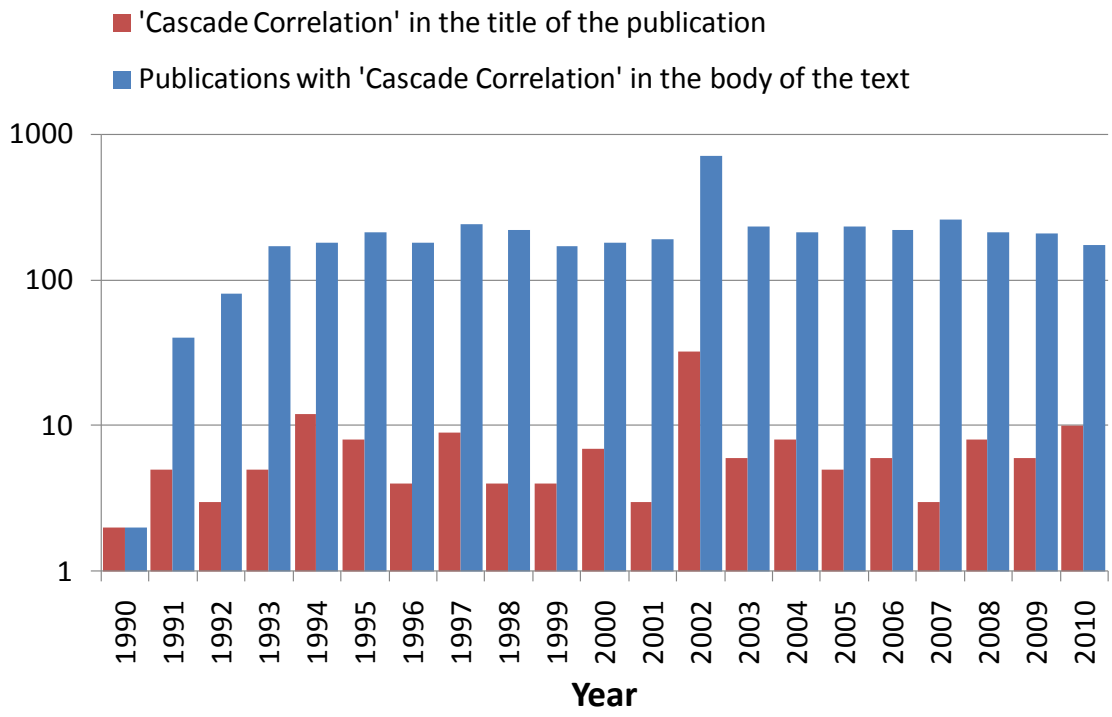



**Figure 2-4 Cascade Correlation neural networks in the literature**

Some articles do exist that illustrate poor performances of Cascade Correlation on benchmark problems such as Banks et al. [28]. However, on further inspection, Banks's trainings have been conducted in the absence of early stopping, ensembling, or any other variance reducing technique (2.6.1) – despite the well known propensity of Cascade Correlation neural networks to lose generalisation due to overfitting [29].

With her implementation of the Cascade Correlation, Schmitz [30] gives a thorough treatment of surrogate modelling with this neural network. In addition, she modifies its training mechanism such that it trains more rapidly as well as integrating the BFGS optimisation algorithm for more optimal selection of weights; potentially improving training outcomes. Particular emphasis is given to validating the performance of Cascade Correlation-based metamodels in approximating high dimensional surfaces. In the automated hydrodynamic shape optimisation of a ship's hull [31], her CasCor surrogate assisted optimisation returned a 34% improvement in the objective function (Lift/Drag

ratio) for the 28 dimensional problem. The success of applying her CasCor-based surrogate is underlined by noting that the comparative 'classical' approach yielded only a 26% improvement *and* required more than five times the CPU time.

There are many reasons why CasCor might be chosen as the basis for a potent surrogate model:

1. Verifiably successful trainings on high dimensional surfaces
2. No need to train n-neural networks to surrogate for n-responses (unlike relevance vector machines)
3. No parameters to be tuned post-training (unlike for support vector machines)
4. No tuning of multiple hyperparameters, unlike the Kriging method
5. Training times are much lower than for other neural networks
6. As CasCor is a constructive type of neural network, we are never faced with the problem of having to determine the correct topology (number of neurons) a priori. This potentially solves the bias problem of neural networks.
7. Several well known variance reduction techniques (2.6.1) are available to reduce errors, and therefore improve the fit, of neural networks

One weakness of CasCor is that this neural network can only train off-line (batch learning). However, it is a neural network that trains significantly quicker that most, and the desirable feature of on-line learning could conceivably be instigated by complete re-trainings as new learning samples arrive.

## 2.6 Improving the fit of a Cascade Correlation surrogate

The error present after a neural network has trained on a set of data is composed of three terms:

$$\text{Error} = \text{Variance} + \text{Bias} + \text{Noise}$$

Contrasting with physical experiments, results derived from deterministic computer experiments (i.e. fluids and solids solvers) are not subject to random errors [6]. Hence, when we chose to build a surrogate for such a computer model we need not address the problem of reducing noise.

The remaining *variance* and *bias* are the two error terms to be minimised. If we can minimise both variance and bias then we have maximised the accuracy of the fit of our surrogate model to the response surface of the problem. Variance and bias are equivalently known as over fitting and under fitting as shown in the simplified illustrations Figure 2-2 and Figure 2-3.

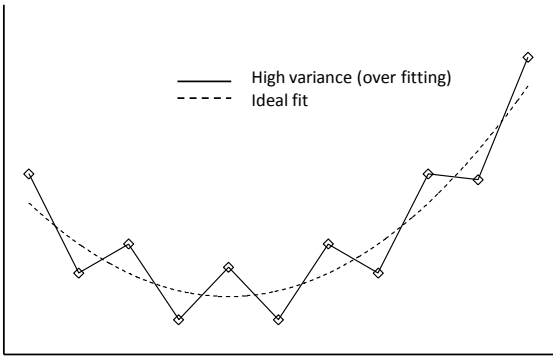



**Figure 2-5 Illustration of Over fitting**

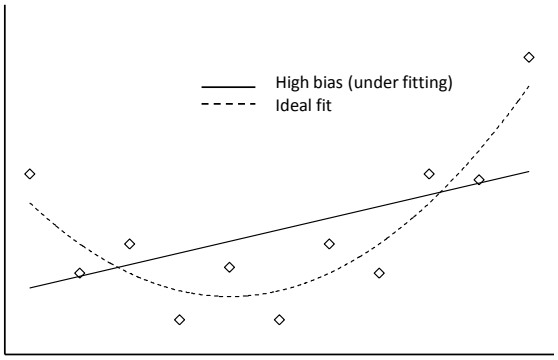



**Figure 2-6 Illustration of Under fitting**

### 2.6.1 Variance reduction methods

Several methods exist in the literature for the treatment of variance.

- Early stopping

  As neural networks train, their error begins to fall as they fit to the underlying function. However, overfitting can occur with too many training epochs. By querying the (still training) neural network with a smaller, unseen set of data, this overtraining can be halted before it has a detrimental effect on the error. Halting the training in this way is known as early stopping. This technique also has the advantage of reducing training times. In determining an early stopping point, the criteria used for this work is presented in section 3.2.4.

- Jitter

  Jitter is the process of deliberately adding artificial noise to the input training data; the output responses are left unchanged. This has the effect of adding new training examples to the training dataset, and acts to improve generalisation for smooth functions when one only has access to a small training set. The noise distribution is assumed to have zero mean and finite variance. Unfortunately, adding jitter significantly increases the training time, which can be impractical for large dimensional training problems [32].

- Weight decay

  A weight decay mechanism assigns larger magnitudes to important weights and smaller values to unimportant weights [33]. Overall the weights decrease, but weights that contribute greater reductions to the training error are reinforced [34]. In [35], this method is described as one that can give excellent generalisation results as CasCor networks grow during training. Schmitz [30], however, discusses the difficulties with implementing weight decay with CasCor networks. The need to tune the different decay constants for the input, output, and hidden layers would add complexity to, and increase the time of, the training.

- Ensembling

  Also known as *a committee of machines*, *bootstrap aggregating*, or *bagging*, ensembling involves training multiple neural networks on the same dataset. In use, the arithmetic mean of their combined response is taken as the response of the ensemble. Due to the mean value smoothing individual variance errors, the response from the ensemble is more accurate than the response of any member of that ensemble. Its disadvantage is the increase in training time imposed by the requirement to train multiple networks.

For this work, jitter and weight decay are disregarded for the following reasons:

- The number of jittered cases to be added, and the selected variance can result in different training outcomes [30]
- Jitter increases training times
- The generalisation of the neural network is very sensitive to the *decay constant* (when using weight decay) – and its calculation is known to be computationally intensive [24].

Early stopping and ensembling are the two methods chosen to reduce variance. Experiments with these two techniques are performed in Chapter 3.

## 2.7 Research Gaps for the Cascade Correlation neural network

Based on a rigorous statistical analysis for the optimal size of an early stopping dataset, Amari [36] suggests the following size:

$$\text{Samples in Early-stop dataset} = 1/\sqrt{2M} \tag{2-1}$$

where M is the number of samples in the training dataset.

As the statistical analysis on which this is based considers fixed-topology neural network types, a research gap exists to explore how differing sizes of early stopping sets influence the reduction in error for the growing topology neural network of CasCor. This gap is explored in section 3.2.4.

Common practice is to ear-mark small *validation datasets* for the purpose of early stopping ([31] [7.5%-30%], [37] [10%-25%], [38] [20%-37%] as percentages of the training dataset). This is because the early stopping set must be formed from samples wholly independent from the training dataset; the larger the early stopping set, the more training samples we will be excluding - to the detriment of successful learning. However, if the success of the training is then to be measured, a larger *testing set* (or *generalisation set*) is typically used. In this way, both the bias and the variance remaining in the model can be evaluated. An alternative approach is explored as part of Chapter 3 that uses a

larger early stopping dataset (>40%) as a proxy for a testing dataset, yet still retains a measure for the bias and variance of neural network.

A further research gap would be to see if the following question can be answered: how much training data does the CasCor neural network need to perform a successful mapping of an arbitrary function? Furthermore, can the demand for training data be expressed in terms of the dimensions of a problem? These gaps are investigated in the work of Chapter 3**.**

## 2.8 Design optimisation

There are many questions to be answered for a researcher, or a research team, that wishes to conduct an exploration of the performance of new, or existing designs. Workflows often differ depending on the problem at hand, but desirable feature are bulleted below. In the late 1980's and the early 1990's, no software suites existed that provided a comprehensive set of features. Those packages that did exist were embryonic - characterised by an incomplete suite of tools and/or poor integration with other packages. Any missing feature required analysis outside of any given software tool; necessitating either the laborious hand-coding of that feature, or the use of several, mutually incompatible, pieces of software. For example, the first version of the Linux-based optimisation toolkit, "Nimrod", in 1995 allowed for a search of the design space via a parameter sweep but it was 2001 before optimisations within the search space became possible with the introduction of Nimrod/O. The Nimrod toolkit is a software suite developed by academics at the University of Monash in Melbourne, but the picture is similar for commercial software. Dassault Systèmes has developed the CAD software, CATIA, since 1981 but it was the 2008 acquisition of Engineous Software that enabled them to incorporate DoE, multi-objective optimisation, and the automation of simulations (via "Simulia") into their Product Lifecycle Management software (PLM).

## 2.9 Optimisation toolkits

As the research field of design optimisation is very mature, many software tools, both commercial and open source, now exist in various forms to enhance the productivity of the engineer:

**Table 2-3 Optimisation toolkits**

| | |
|---|---|
| PHX ModelCentre | http://www.phoenix-int.com/ |
| iSight and Fiper | http://www.simulia.com/ |
| Nimrod | http://messagelab.monash.edu.au/Nimrod/AllOnOnePage |
| Geodise | http://www.geodise.org/ |
| Dakota | http://dakota.sandia.gov/ |
| Technosoft | http://www.technosoft.com/ |

Desirable features for any optimisation toolkit include, but are not limited to the following:

1. An ability to parallelise, and thereby accelerate, optimisation jobs (The workload is either shared between the cores of multi core CPUs or distributed on a local/wide area network for objective function evaluations on multiple machines such as a Grid, Cloud, or Cluster of computers)
2. Job dispatch, control, and error reporting
3. Tools to assist with sampling of a search space (DoE)
4. Work flow management
5. In-built optimisation algorithms
6. Surrogate modelling

At the commencement of the current work in 2007, the Nimrod team had not released the DoE and workflow-management modules (Nimrod E and K) for public download. The difficulties of workflow management highlighted in

Chapter 5 and the lack of a DoE module at first pointed to these as two research gaps worthy of investigation. At time of writing, those gaps have been filled by work of the Monash team. In 2009 two existent research gaps were; the lack of a built-in surrogate model and the lack of a multi-objective optimisation algorithm in Nimrod/O (points 5 and 6, above).

## 2.10 Multi-objective optimisation

Mathematical optimisation techniques have existed since the 18th century when Newton, Euler and Lagrange used the calculus of variation to develop methods for evaluating minima and maxima of differentiable functions, however it was only when Pareto developed his theory of optimality that a framework existed for multi-objective optimisation problems (MOOP) [39]. When two or more objectives are to be optimised simultaneously, a true multi-objective optimisation process will not reduce to a single ideal solution if any of the objectives are in conflict with each other. This is the case with the aerofoil optimisation of Chapter 5; high lifting wings tend not to be associated with low drag co-efficients.

The defining characteristic of the Pareto optimal set is that one objective function can only be improved if at least one other objective function is degraded. A multi-objective optimisation algorithm searches for the *Pareto front*. The minimisation of a general two criteria multi-objective optimisation is formulated as follows:

Minimise $f(\mathrm{x}) = (f_1(\mathrm{x}), f_2(\mathrm{x}))$ such that $\mathrm{x} \in \mathrm{X}$, the feasible region

subject to $\begin{cases} g_j(\mathrm{x}) = 0 & j = 1,...,M \\ h_k(\mathrm{x}) = 0 & k = 1,...,K \end{cases}$ constraints

where $\mathrm{x}$ is a *p*-dimensional vector whose components are known as decision variables, $g_j$ are equality constraints and $h_k$ are inequality constraints.

Definition of *dominance*: Comparing two solutions, $x_1$ and $x_2$, we say that $x_1$ dominates $x_2$ if:

$$\left[ f_1(x_1) < f_1(x_2) \text{ and } f_2(x_1) \leq f_2(x_2) \right] \text{or} \left[ f_1(x_1) \leq f_1(x_2) \text{ and } f_2(x_1) < f_2(x_2) \right] \quad \textbf{(2-2)}$$

The *Pareto set* is formed from only those solutions that are not dominated by any other (i.e. from *non-dominated solutions).* The *Pareto front* is an imaginary construct in the objective space, along which candidates from the *Pareto set* would lie.

Since the 1980's, sufficient computing power has existed to approach the MOOP via the use of bio-inspired metaheuristics. The focus of optimisation has shifted from mathematical programming techniques to the application of evolutionary methods, which adapt the genes of a population of candidates with the aim of improving their "fitness". Mathematical programming techniques, in general, generate one element of a Pareto set and are susceptible to changes in the shape of the Pareto front and may not work when this front is non-convex and/or discontinuous [40]. By contrast, population-based evolutionary algorithms simultaneously manipulate a set of possible solutions. In addition, evolutionary algorithms are more robust to discontinuous or non-convex Pareto fronts [40]. For this reason, they are known as "robust" optimisation methods. Examples include: Strength Pareto Evolutionary Algorithm (SPEA)[41], Non-dominated Sorting Genetic Algorithm (NSGA)[42], Multi-Objective Tabu Search (MOTS)[43], and Differential Evolution for Multiobjective Optimization (DEMO) [44]. The current author contributes by interfacing DEMO to Nimrod/O (Chapter 6).

# 3 EARLY STOPPING AND ENSEMBLING FOR THE VARIANCE PROBLEM OF CASCADE CORRELATION NEURAL NETWORKS

## 3.1 Introduction

The aim of the work in this chapter is to investigate the use of early stopping and ensembling to reduce variance errors, Figure 3-1 thereby improving the fit of the neural network to the underlying functions.

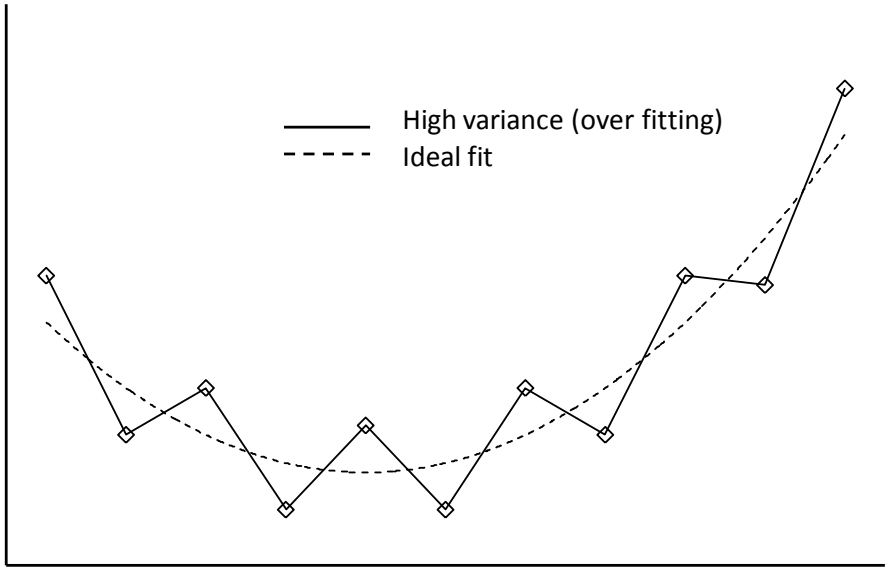



**Figure 3-1 High variance**

The objectives of this chapter are to investigate the effects of: the size of the early stopping dataset, the size of the training dataset, how the demand for training data varies with the dimensions of the problem, whether a testing dataset is strictly necessary, and the limitations of early stopping and ensembling.

As the intended use of Cascade Correlation (CasCor) is to create a metamodel to assist with design optimisation, multimodal test functions for global optimisation [45] (typically employed to test optimisation algorithms) offer

appropriate surfaces upon which to test the metamodel. These test functions are used for the work in this chapter and for the work in Chapter 4, hence they are tabulated in the appendix (Table A-1). All the test functions used are smooth and continuous and no noise is present in (or added to) any of the datasets. Many of the test functions used are multimodal and, prior to the modifications made by the Thesis, were chosen because they posed significant mapping problems for this neural network.

### 3.1.1 Early stopping

One of the disadvantages of CasCor neural networks is their propensity to overfit on the training data, thus decreasing the quality of the approximated fit of the underlying function [29]. Inspecting the monotone decrease of the training mean squared error (MSE) gives no indication of this. Typically, the error during training is seen to reduce, almost uninterrupted, until one of the stopping criteria is met and the network is pronounced as “trained”. If, however, a call-back function is set, the training progress can briefly be interrupted to test the (still evolving) neural network against the validation dataset. A call-back function is useful for customising any training procedures and is implemented as part of the neural network library used in this work [46].

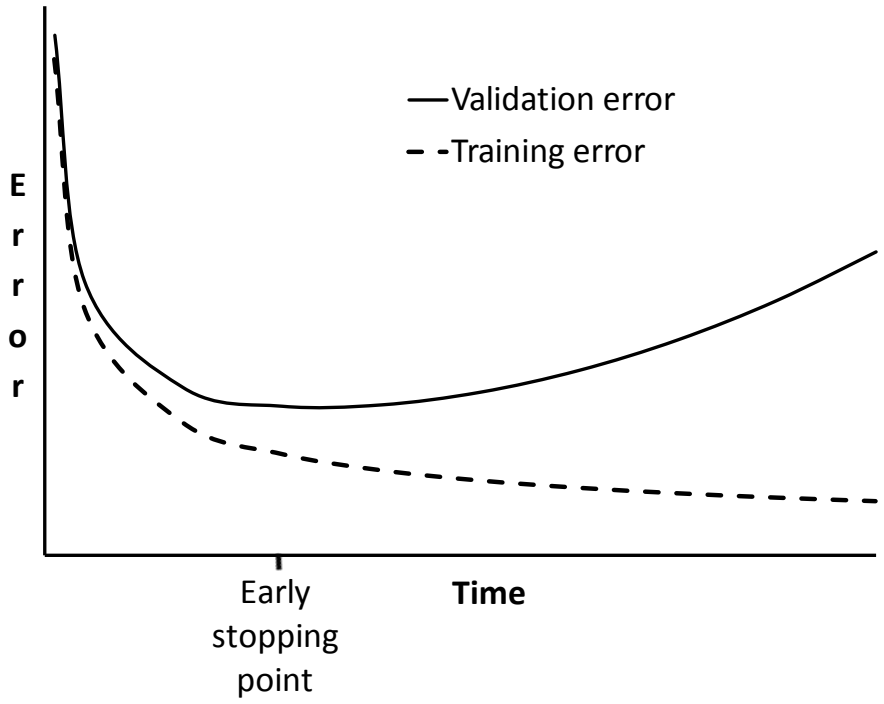



**Figure 3-2 Early stopping with a validation dataset**

The validation dataset is wholly independent from the training set and it allows us to determine an early stopping point. The MSE graph on this validation data typically takes the approximate form of a hockey stick outline – initially the validation MSE falls as the network fits to the underlying function but, at some point, too many neurons are added and any gains from a reduction in bias become off set by a disproportional increase in variance of the neural network. After this point, any more training acts to further increase this variance, resulting in a net increase in the overall MSE (Figure 3-2). Early stopping halts the training at or around this minimum point thus minimising negative impacts from overfitting. In reality, the profile of the validation error is not smooth and some form of heuristic needs to be used to halt the training at an appropriate moment; the heuristic introduced by this author is described in 3.2.4.

### 3.1.2 Ensembling

Tetko and Villa [29] described ensemble averaging, or a “committee of machines”, as acting to reduce the variance error. It is likely that each neural

network in that committee will have approximated the response surface of the training data differently due to the random initialisation of their initial weight values and the non-deterministic nature of neural network training. Ensembling smoothes the responses of its members in the following way: multiple neural networks are trained on the same dataset, but in use, the arithmetic mean is taken across the output responses of the ensemble members. When compared to the basic CasCor neural network, the testing error of these ensembles is much lower than the average test errors of their constituent parts - the only penalty being an increase in required training time.

## 3.2 Experimental set up

The architecture of the CasCor algorithm is well known [25, 47, 48]. The CasCor neural networks under consideration are created from the open source library created by Nissen [46]. The library contains an implementation of the Cascade Correlation II algorithm based on the original Lisp code written by Fahlman in 1996 (unpublished).

Here, the FANN C source code is used with default settings chosen for CasCor training. The target MSE for the training is $10^{-4}$ when early stopping is not used and a nominal setting of $10^{-5}$ when early stopping is used. In use, the lower target would only be reached for trivial test cases. More likely is that early stopping will trigger a halt to the training before the training error reaches $10^{-5}$. The existing release, 2.1.0-Beta, does not provide a neural network copy utility or functions that correctly scale and de-scale datasets, and so these have been added to this author's implementation.

### 3.2.1 Training datasets

The training datasets consist of repeated runs of $OA.16.5.4.2$ [49]. With 16 evaluations being made each time, 6 runs of this OA will be required to generate a training dataset of 96 points. This OA allows for a design of experiments in up to five dimensions. For test functions in less than five

dimensions, the OA is trimmed by removing unneeded columns. This does not affect the orthogonal properties of the array. The selection of the factors in each subsequent OA is known as the infill criteria [50]; when subsequent OAs are evaluated, each of its factors is chosen to be numerically furthest from all previously tested factors.

### 3.2.2 Testing the fit

One traditional test for the quality of regression fits (such as presented in the current work) is to calculate the MSE against a testing set, in which the samples differ from those in the training set. Lower is better, and so we can measure the success of the techniques herein by how much they reduce the MSE. The testing sets are generated from the algorithm in [51]. The size is chosen as $1000 \times d$ where $d$ is the number of inputs to the neural network (or dimensions). The positioning of so many points is computationally expensive, especially when trying to maintain space filling properties. For this reason only one template was generated for each of the four different dimensions that were tested.

The range of all inputs and outputs is normalised to the interval [0.1,0.9] with the scaling factors saved after processing. These factors are later used to scale down the queries and scale up the neural network response.

Note: Unless otherwise stated, the MSE errors presented in this chapter are calculated on scaled data [0.1,0.9], thus making possible fair comparisons between otherwise disparate function output ranges.

### 3.2.3 Sample size

When choosing the size of the training datasets, how many samples should be used? Too few samples will mean that the training set may not accurately represent the underlying pattern. However, in situations where generating training data is very time-expensive, it would be useful to know the minimum size that can be of practical use when training CasCor neural networks. Another

question to be answered is; how the demand for training data varies with the dimensions of the problem at hand? To determine the answers to these questions CasCor training was performed using 13 test functions (defined in Table A-1) in two, three, four and five dimensions with training datasets sizes in the range $[16 \times d, 384 \times d]$ (where $d$ is the number of dimensions).

### 3.2.4 Early stopping

Several tests were undertaken in order to answer two questions: 1. What is the smallest size of validation set that can be used? 2. Does the use of larger size validation sets have any beneficial effect on improving the fit of the trained networks? The validation sets ranged from a size of 5% of the training set to 100% of the training set. Code from Beachkofski and Grandhi [51] provides the method of distributing the samples in the validation set. This “improved Latin hypercube” sampling was chosen because:

1) Generating validation sets of less than 1000 points is not computationally expensive and can be done at run time,

2) The algorithm in [51] produces points that fill the hypercube uniformly, the statistical properties of which are desirable as described in [50],

3) The technique is fundamentally different from that used to generate the training set - ensuring that most, if not all, of the validation data points are automatically independent from those in the training set.

After the validation error is initialised to 1.0, this author’s heuristic algorithm for early stopping is run each time a new hidden neuron is added to the network, and is given below:

- Test the network against the validation set.
- If this new validation error is less than the old one, update the old validation error with this new value and make a copy of this “best network so far”.
- Do not initiate early stopping until at least five hidden neurons exist in the

network.

- Trigger early stopping on the earliest of:
  - The error on the validation set becoming less than $5 \times 10^{-5}$ (suitably low error)
  - The validation error growing to be 50% larger than the smallest experienced validation error (network is diverging)
  - More than 31 hidden neurons existing in the network (likelihood of a diverging network)
- When early stopping occurs, the "best network so far" is recalled from memory to replace the active network. The training is halted and the network is saved to permanent storage. The saved neural network is therefore that which had the smallest validation error.

### 3.2.5 Dispensing with the testing set

Early stopping validation sets share the same property of a testing set in that they both contain samples wholly independent from the training dataset. The only difference is that testing sets are usually of a large size. Testing sets are useful in determining how successful a neural network's training has been. However, in the case of surrogate modelling, sampling for datasets is likely to be very time-expensive. If we want to avoid the cost of generating a large testing set, yet still retain a test for the quality of the fit, is there a size of validation set that can give us a reasonable approximation to the results we would get from a testing set? Experiments were performed that compare the MSE calculated from validation sets of sizes [5%,100%] of the training set against MSE calculations from much larger testing sets of size $1000 \times d$.

### 3.2.6 Ensembling

When preparing an ensemble, we need to answer the question of how many neural networks we should include in that ensemble. Others have chosen an

arbitrary number [29, 32] for their ensembles, but here the ensemble size is investigated with respect to its influence on reducing the MSE.

Ensembles of CasCor neural networks were trained on the 13 test functions (Table A-1); each test was repeated ten times for the larger ensembles and 30 times for ensembles smaller than ten.

## 3.3 Results

### 3.3.1 Sample size

Figure 3-3 shows the results of the sample size test of training on the 13 test functions in the appendix, covering two to five dimensions. Each test was repeated ten times. After each training, the quality of the fit was evaluated by a testing set of size $1000 \times d$. The resulting MSEs often differed by one or two orders of magnitude, hence a need to normalise the results. In normalising the results, the mean squared errors for each function were scaled such that the size of the training dataset that yielded the worst error was attributed 1.0; the training dataset set size that gave the lowest MSE was attributed a score of 0.0.

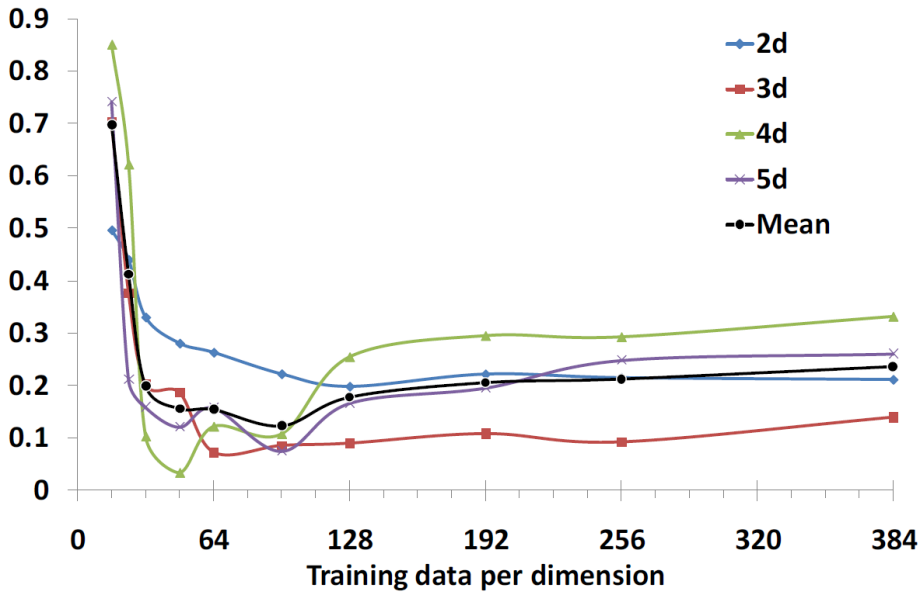



**Figure 3-3 Change in testing MSE against training set size**

Figure 3-3 shows the mean of the normalised error per dimension and also the mean of all 13 test functions. A contribution of the current work is the experimental finding that CasCor's demand for training data scales linearly with the number of dimensions, and is not correlated with the nature of the surface of the test function. Instead, the demand for training data is directly proportional to the total number of weights that the training algorithm is required to optimise.

In all cases, less than 32 samples/dimension are seen to lead to poor mappings of the underlying function. This corroborates Schmitz's success with her 28 dimensional CasCor metamodel; her 1000 samples (i.e. 35.7 samples/dimension) was a sufficient, but not ideal, sized training set [31] .

For this work, optimal training occurred when the training datasets were between the sizes of 48 and 128 samples/dimension.

### 3.3.2 Early stopping

Figure 3-4 shows the results of an experiment to determine how big the validation set should be with respect to the training set. For this experiment, 96 samples /dimension was chosen as the training set size. As before, training was conducted on all 13 test functions and each test was repeated ten times; Figure 3-4 shows the mean average of the results.

A logarithmic trend line has been fitted to the data points in Figure 3-4 that shows the error reducing by 25% as the size of the validation set is increased from 5% to 100% of the training set. However, the conclusion drawn here is that validation set sizes as small as 5% (or minimum size of 10 samples) could be relied on to achieve much of the desired early stopping effect.

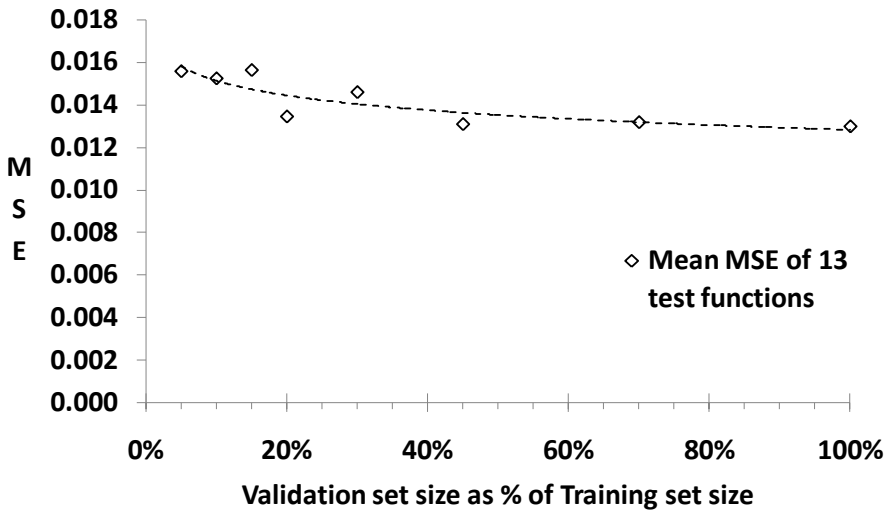



**Figure 3-4 Reductions in the tested MSE with larger early stopping/validation set sizes**

In Table 3-1, the results of early stopping are displayed. For all the experiments in this table, the training datasets were created from 48 samples per dimension and the validation sets were set at 20% of the size of the training datasets. The mean reductions in the MSE range from 8% to 57% due to early stopping (ES). In all test cases, early stopping has reduced the common tendency of the CasCor neural network to overfit.

### 3.3.3 Dispensing with a testing set

There was one other early stopping experiment for which we desired an answer. If an unseen dataset is used for the early stopping set, then can we dispense with a testing set entirely – relying only on the MSE calculated from the validation dataset? If this approach is viable then, in circumstances when creating datasets is time-expensive, we could dispense with the creation of a testing set - relying solely on the validation error as a test for the quality of fit.

The results in Figure 3-5 were generated from the same experiment performed for the results in Figure 3-4. However, for each size of validation set, the MSE calculated from the validation set was compared to the MSE calculated from the

much larger testing sets $(1000 \times d)$. The validation and testing sets only differ in the number of samples; both share the same property of containing samples independent from those in the training dataset. The results suggest that validation sets of 20% or greater are sufficient to give a close approximation to the results from a much larger testing set. Taking a two dimensional test function as an example; the training set would have numbered $48 \times 2 = 96$ samples, and a 45% validation set would have been of size $96 \times 0.45 = 44$. The total number of samples we would have created $= 140$. With this validation set, Figure 3-5 predicts that the MSE calculated from this, size $= 44$, validation set will be within 7% ($\sigma$ = 5%) of the MSE calculated from a testing set of size $= 2000$ samples. This represents a significant time saving; if each sample costs, for example, 20 minutes to generate, we save 25 days of sampling.

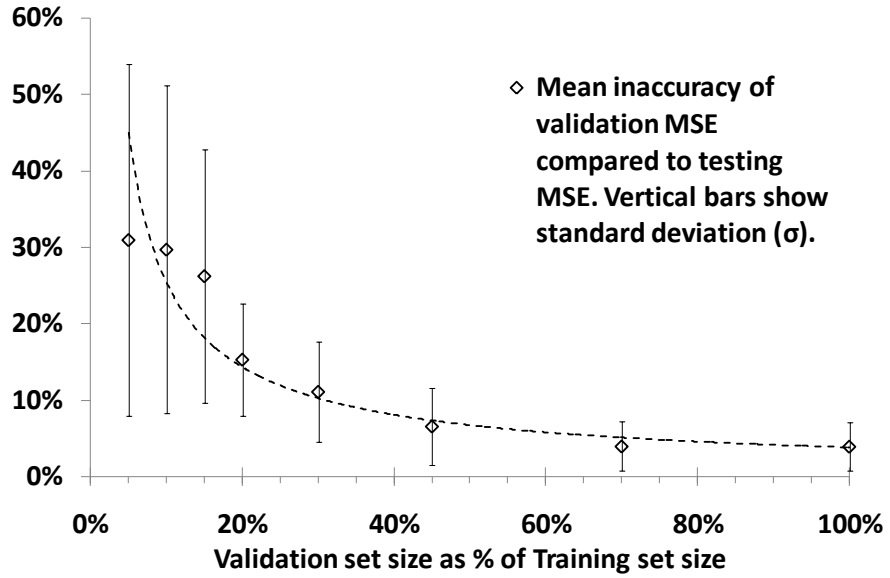



**Figure 3-5 How close the validation dataset MSE is to the MSE from the testing dataset**

### 3.3.4 Ensembling with Early Stopping

For clarity, only three of the thirteen test function errors are shown in Figure 3-6, however, the form of the line graphs were similar throughout all 13 functions; the MSE reduced rapidly as the ensemble size increased from one to seven. Smaller reductions in the MSE occurred until ensembles with a size greater

than 25 were seen to deliver little benefit. Early stopping was also applied for this experiment and so the MSEs in Figure 3-6 reflect the combination of both techniques.

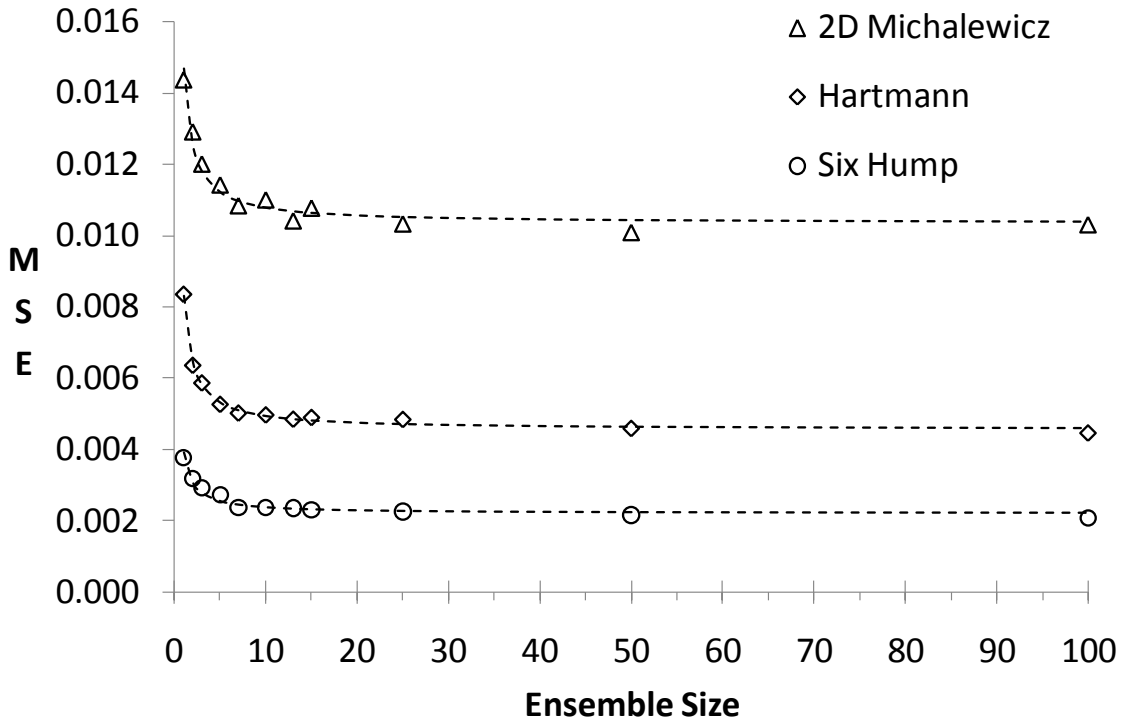



**Figure 3-6 Reductions in MSE due to ensembling**

Shown in Table 3-1 are the quantitative results of applying early stopping, and, early stopping combined with ensembling. Across all test functions, the mean squared error is reduced by a factor of 2.8 by a combination of both techniques.

**Table 3-1 Benefits of early stopping (ES) and ensembling (Ens)**

| | | | MSE $\times 10^3$ | | |
|---|---|---|---|---|---|
| Test function | Dims | Size of train + early-stop sets | Cascade Correlation (CasCor) | **CasCor + ES** | **CasCor with Ens + ES** |
| Ackley | 2 | 116 | 33.79 | 14.33 | 3.10 |
| | | | Reduction in error: | 57.59% | 90.82% |
| DeJongs5th | 2 | 116 | 176.33 | 80.06 | 58.10 |
| | | | Reduction in error: | 54.60% | 67.05% |
| Langermann | 2 | 116 | 77.33 | 33.32 | 22.43 |
| | | | Reduction in error: | 56.91% | 70.99% |
| Michalewicz | 2 | 116 | 22.90 | 14.38 | 10.78 |
| | | | Reduction in error: | 37.22% | 52.92% |
| Schwefel | 2 | 116 | 36.73 | 19.96 | 4.39 |
| | | | Reduction in error: | 45.67% | 88.06% |
| Shubert | 2 | 116 | 32.08 | 20.24 | 4.59 |
| | | | Reduction in error: | 36.89% | 85.69% |
| Six Hump | 2 | 116 | 13.39 | 6.77 | 4.26 |
| | | | Reduction in error: | 49.42% | 68.15% |
| Ackley | 3 | 173 | 14.66 | 6.36 | 5.64 |
| | | | Reduction in error: | 56.62% | 61.56% |
| Hartmann | 3 | 173 | 12.67 | 11.60 | 6.50 |
| | | | Reduction in error: | 8.40% | 48.66% |
| Rosenbrock | 4 | 231 | 18.27 | 14.41 | 8.19 |
| | | | Reduction in error: | 21.10% | 55.18% |
| Schwefel | 4 | 231 | 27.47 | 20.73 | 13.70 |
| | | | Reduction in error: | 24.51% | 50.12% |
| Michalewicz | 5 | 288 | 10.64 | 9.52 | 5.38 |
| | | | Reduction in error: | 10.53% | 49.45% |
| Schwefel | 5 | 288 | 44.77 | 22.61 | 22.07 |
| | | | Reduction in error: | 49.50% | 50.71% |
| **Average reduction in error** | | | | **39.15%** | **64.57%** |

### 3.3.5 Discussion

The curves in Figure 3-6 take the form:

$$MSE_{Ensemble} = \frac{\left(\overline{MSE}_{EnsSize=1}\right) - Bias^2}{EnsSize} + Bias^2 \qquad \textbf{(3-1)}$$

where $\overline{MSE}_{EnsSize=1}$ is the mean MSE of the neural networks that constitute the ensemble. $Bias^2$ is the asymptote to which the curves tend. Effectively, the bias

is an MSE boundary that no size of ensemble can reduce because ensembling acts only on the part of the error that is due to variance. Likewise, early stopping, provided by the validation set, acts only to reduce the variance by limiting overfitting.

A high bias can be thought of as representing a lack of complexity in the regression model. For example: if a highly multimodal surface is modelled with a low complexity / low modality surface, we would expect to find bias dominating the MSE.

Equation (3-1) can be derived from the equations presented in the seminal paper of Geman et al. [52] where they describe the bias/variance dilemma of neural network training. The general form of the error is given in their paper as:

$$Error = Variance + Bias^2 \quad \textbf{(3-2)}$$

and it can be shown that (3-1) and (3-2) are equivalent. Equation (3-1) provides a convenient test for the relative contribution of variance and bias to the overall error. Evaluating the MSE is a function commonly built into neural network libraries and so, using MSE evaluations alone, new formulae are presented here for estimating the bias (3-3) and then the variance (3-4) for any ensemble. These are, to the author's knowledge, new formulations for determining variance and bias. In appendix B.1, this new method is compared and contrasted to Geman's method for finding bias and variance.

$$Bias^2 = \frac{(EnsSize \times MSE_{Ensemble}) - \overline{MSE}_{EnsSize=1}}{(EnsSize - 1)} \quad \textbf{(3-3)}$$

$$Variance = \frac{\left(\overline{MSE}_{EnsSize=1}\right) - Bias^2}{EnsSize} \quad \textbf{(3-4)}$$

If variance is found to dominate, then creating a larger size of ensemble will reduce the MSE and improve the mapping of the underlying function. If we find that the bias is the largest component of our mapping error, we know that the information capacity of our CasCor neural network has been exceeded.

Installing more neurons will confer additional capacity: in Chapter 4, this author introduces ‘patchworking’ to achieve an increase in capacity.

By way of example, Figure 3-7 presents a smaller region of Figure 3-6 and, for clarity, only the Michalewicz data is re-plotted. Say that an ensemble of size 10 has been created. We calculate the MSE of that ensemble and also calculate the mean MSE of the 10 members of that ensemble.

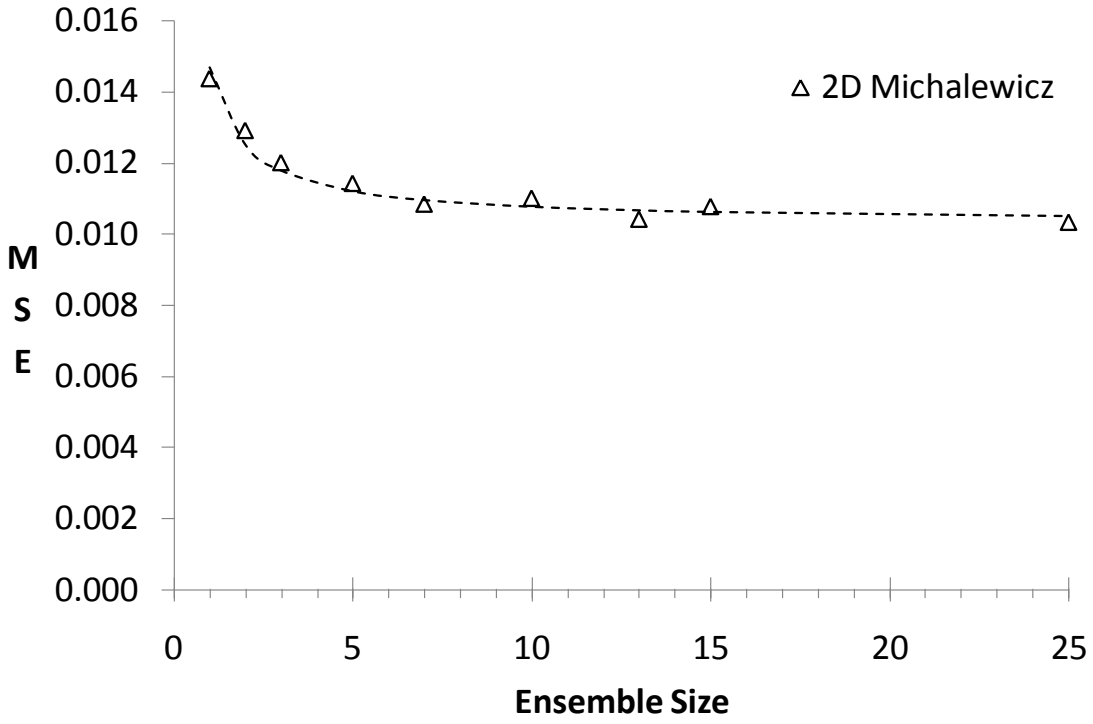



**Figure 3-7 Reductions in MSE due to ensembling (Michalewicz data replotted)**

Now, by using (3-3) and (3-4), we find that our $Bias^2$ = 0.01 and the *Variance* of our ensemble = 0.0004. Ensembling to a size of 15 would reduce our variance to 0.00027, but it is clear that the dominant component of our MSE is the bias. A CasCor ensemble that possesses a high bias indicates a highly multimodal function in the training dataset. When the MSE is undesirably high (and dominated by bias), the application of this author’s patchworking method is advocated (Chapter 4).

### 3.3.6 Visualisation of the benefits of early stopping and ensembling

In Figure 3-8, the Six Hump Camel Back test function is displayed. The result of training a single CasCor neural network (Figure 3-9) shows high variance. The mapping is improved with the application of early stopping (Figure 3-10) and significantly improved by an ensemble of early stopped networks (Figure 3-11).

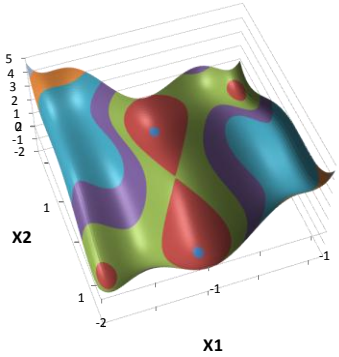



**Figure 3-8 Six hump test function**

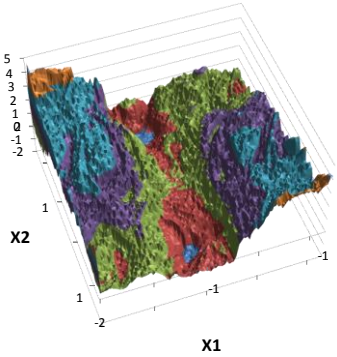



**Figure 3-9 CasCor's high variance**

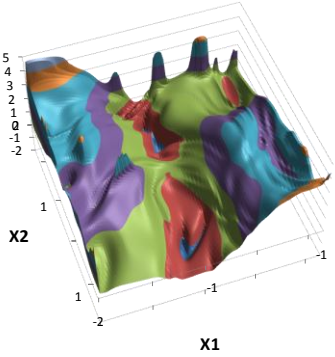



**Figure 3-10 With early stopping**

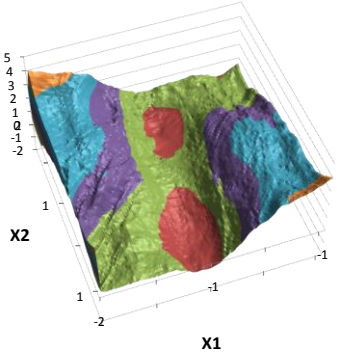



**Figure 3-11 With ensembling and early stopping**

### 3.3.7 Qualitative evaluation of CasCor training

Based on the experiments conducted as part of this thesis, it has been possible to create a table (Table 3-2) that qualitatively describes how successful a neural network's training is likely to have been.

**Table 3-2 Qualitative evaluation of CasCor training**

| Neural network error; testing dataset | Quality of mapping | Comments |
|---|---|---|
| $MSE > 5\times10^{-2}$ | No real mapping | No mapping has been found that represents the features in the dataset. |
| $5\times10^{-2} > MSE > 1\times10^{-2}$ | Very poor | Some patterns were found in the dataset. The neural network has made an approximation to those patterns, albeit poorly. |
| $1\times10^{-2} > MSE > 5\times10^{-3}$ | Poor | The underlying function has not been mapped in detail. Predictions from this neural network should be made with caution. |
| $5\times10^{-3} > MSE > 1\times10^{-3}$ | Good | The underlying function has been mapped quite well but not precisely. Fine details of the response surface will not have been captured. |
| $1\times10^{-3} > MSE > 1\times10^{-4}$ | Very good | A successful mapping. Predictions from the neural network can be made with confidence. |
| $MSE < 1\times10^{-4}$ | Excellent | An almost perfect mapping; the fine details of the features in the training dataset have been captured accurately. |

## 3.4 Usage with real world data

The datasets in this chapter have thus far been generated from mathematical test functions. The highly multimodal functions have in some cases caused exceptional mapping problems for the CasCor neural network, despite the application of early stopping and ensembling methods. However, are problems in the real world as complex as some of these mathematical functions? The answer to that question is “it depends”. We have seen that the demand for training data varies linearly with the number of dimensions so, in the case where we have (or can generate) sufficient samples for our train and test datasets, the only factor that will preclude us from accurately modelling a real-world problem is the modality of the response surface. If our real world training data describes a highly multimodal surface then ensembling and early stopping will not be sufficient tools for us to map the problem. Conversely, if the modality of the response surface is low, we will be able to build a useful CasCor metamodel. Two real world examples follow; Concrete Compressive Strength (eight dimensional), and Abalone Age Testing (eight dimensional)

### 3.4.1 Concrete Compressive Strength

The owner of this dataset is Prof. I-Cheng Yeh of Chung-Hua University in Taiwan. He has donated this dataset for public use and it is available on the machine learning repository website [53]. He has published six papers on the subject of concrete compressive strength, the first in 1998 [54] and the latest in 2006 [55]. He describes the compressive strength of concrete as a highly nonlinear function of its ingredients and its age. The dataset provided has 1030 samples generated from laboratory experiments. All the data is quantitative, with eight input dimensions and one output; the compressive strength measured in MPa. The dataset is provided unscaled, the nine variables are tabulated below in Table 3-3.

**Table 3-3 Concrete compressive strength input and output data**

| Component | Units |
|---|---|
| Cement | $Kg/m^3$ in a mixture |
| Blast Furnace Slag | $Kg/m^3$ in a mixture |
| Fly Ash | $Kg/m^3$ in a mixture |
| Water | $Kg/m^3$ in a mixture |
| Superplasticizer | $Kg/m^3$ in a mixture |
| Coarse Aggregate | $Kg/m^3$ in a mixture |
| Fine Aggregate | $Kg/m^3$ in a mixture |
| Age | Days |
| Concrete compressive strength | MPa (Output Variable) |

Taking this dataset, an early stopping/validation set of size 309 set was first created by separating a randomly chosen 30% from the total of 1030, hence a training set of size 721 remained for training a CasCor based metamodel. This size of the training set represents 90 samples/dimension which is well above the minimum threshold of 32 samples/dimension calculated earlier in this chapter. An ensemble size of 15 was chosen to minimise the inevitable variance. There were no other choices necessary before commencing CasCor neural network training. Training time was approximately 15 minutes on a Laptop (CPU=Pentium SU4100 1.3GHz) i.e. ~ $8.5 \times 10^6$ MFLOP. This contrasts with Yeh's training time of 30 seconds for the neural network he ultimately chose to use for this dataset. Yeh's neural network comprised one hidden layer with four neurons. He did not use the CasCor type of network, and in his paper he described that he made this choice only after performing a number of trials to choose the optimal topology of his neural network and to tune the training parameters.

### 3.4.2 Concrete strength results

Yeh gives the testing error of his neural network as 4.32 MPa RMS (root mean squared on upscaled data). For this author's comparative test, the 309 sample validation set (as unseen data) was also employed as a testing set for the CasCor ensemble. The trained ensemble of the current author had a mean of 6.27 neurons performing the mapping of this problem. The testing error was 4.09Mpa RMS ($MSE = 2.87 \times 10^{-3}$ on scaled data) which represents a 5% improvement over Yeh's neural network. This improvement is made more significant when we consider that no tuning, setup, or specialist knowledge of the problem at-hand was required to achieve this result. For reference, Table 3-4 reproduces an extract of results generated by querying the CasCor ensemble on the validation set.

**Table 3-4 Concrete compressive strength prediction (sample of results)**

| Cement | Blast Furnace Slag | Fly Ash | Water | Superplasticizer | Coarse Aggregate | Fine Aggregate | Age | Compressive Strength, Expected Output (Mpa) | CasCor ensemble Output (Mpa) | Difference (Mpa) | Difference (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 424 | 22 | 132 | 178 | 8.48 | 822 | 750 | 7 | **39.0** | **37.5** | -1.6 | -4.1 |
| 424 | 22 | 132 | 18 | 8.92 | 822 | 750 | 7 | **40.3** | **39.3** | -1.0 | -2.5 |
| 202 | 11 | 141 | 206 | 1.72 | 942 | 801 | 7 | **15.1** | **12.4** | -2.7 | -17.0 |
| 284 | 15 | 141 | 179 | 5.46 | 842 | 801 | 3 | **13.4** | **17.5** | 4.1 | 30.6 |
| 359 | 19 | 141 | 154 | 10.91 | 942 | 801 | 28 | **62.9** | **56.8** | -6.1 | -9.7 |
| 359 | 19 | 141 | 154 | 10.91 | 942 | 801 | 7 | **35.8** | **38.9** | 3.1 | 8.7 |

### 3.4.3 Abalone age prediction

Abalone are edible marine molluscs. Conventional age testing can be described as a time-consuming and boring process that involves cutting the shell, staining the cone, and counting the rings under a microscope; hence the benefits of determining age from more readily measured quantities such as weights and dimensions.

The data here are also found on the machine learning website and derive from a non-machine learning dataset [56] that is used in the scaled form provided by that website. There are 8 input dimensions, of which 7 are numeric and one output dimension (age, years). For the current work, categorical data (Gender) uses the following substitution; Male=1, Female=0 and Infant=0.5. Numeric data that may also be relevant, such as weather and food availability, are not available for this dataset.

The total dataset numbered 4177 samples from which a validation dataset of size 1253 was separated. This validation/early stopping dataset, as unseen data, was also used for final testing. As for the concrete compressive strength training, an ensemble of size 15 was chosen to reduce the inevitable variance. Training time was greater than that for the concrete dataset; approximately double (on the same computer).

### 3.4.4 Abalone age prediction results

A sample extract of the results is presented in Table 3-5. Comparative benchmarks are available in the literature [57]; Neural Network built using ‘R’ [58] MSE = 4.31.The MSE for the CasCor ensemble of this work was 2.3% lower at 4.21, and employed a mean of 6.33 neurons. For an undetermined reason the ages of infants was predicted with a greater accuracy than samples for which gender was known; MSE infants = 2.57, MSE Male = 4.62, MSE female = 5.50.

**Table 3-5 Abalone age prediction (sample of results)**

| Gender | Length | Diameter | Height | Whole Weight | Shucked Weight | Visceral Weight | Shell Weight | Actual Age, Expected Output (Years) | CasCor ensemble output (years) | Difference (Years) | Difference (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.51 | 0.40 | 0.14 | 0.81 | 0.46 | 0.20 | 0.20 | **10.0** | **8.0** | -2.0 | -20 |
| 1 | 0.64 | 0.50 | 0.19 | 1.30 | 0.44 | 0.26 | 0.47 | **16.0** | **16.1** | 0.1 | 0.6 |
| 1 | 0.50 | 0.40 | 0.17 | 0.83 | 0.25 | 0.21 | 0.29 | **13.0** | **14.1** | 1.1 | 8.5 |
| 0 | 0.49 | 0.40 | 0.16 | 0.66 | 0.25 | 0.13 | 0.24 | **14.0** | **13.0** | -1.0 | -7.1 |
| 0.5 | 0.41 | 0.30 | 0.12 | 0.32 | 0.13 | 0.07 | 0.11 | **7.0** | **8.4** | 1.4 | 20 |
| 0 | 0.47 | 0.35 | 0.15 | 0.52 | 0.19 | 0.12 | 0.18 | **11.0** | **11.8** | 0.8 | 7.3 |
| 0.5 | 0.43 | 0.38 | 0.11 | 0.33 | 0.13 | 0.08 | 0.10 | **10.0** | **8.2** | -1.8 | -18 |
| 0.5 | 0.37 | 0.27 | 0.09 | 0.21 | 0.08 | 0.05 | 0.07 | **7.0** | **7.0** | 0.0 | 0 |
| 1 | 0.57 | 0.44 | 0.18 | 0.90 | 0.31 | 0.19 | 0.33 | **14.0** | **14.5** | 0.5 | 3.5 |

### 3.4.5 Summary of real world test data

The purpose of including this test data was primarily to demonstrate examples of the successful application of CasCor neural networks on real, as well as synthetic problems. The examples were sought from the machine learning repository [1]. Of interest is to note that the majority of their test data did not conform to the requirements of the CasCor surrogate of the current work, namely; many datasets were classification, or mixed (regression/classification),

and many examples contained too few samples and so did not conform to the minimum-samples/dimension threshold of 32 per dimension.

Results for the concrete and abalone examples do compare favourably with similar tests found in the literature. Although both CasCor ensembles examples here did return lower errors (either MSE or RMS) than the errors quoted in the literature, it should be stated that the experimental set-ups were not identical. Specifically the methods of generating the training/testing/validation datasets and the method of testing for the final error differed and so a direct performance comparison is not possible.

When the bias and variance of the concrete and of the abalone ensembles was calculated, it was found that bias constituted >97% of the remaining error. To reduce this error, we would require our CasCor ensemble to have a greater information capacity.

## 3.5 Conclusion

The aim of the work in this chapter was to perform experiments with the known techniques of early stopping and ensembling. The initial objectives were to find out what were the effects of: the size of the early stopping dataset, the size of the training dataset, how the demand for training data varied with the number of dimensions, and whether a testing dataset is strictly necessary. These objectives have been achieved and constitute contributions of the current work.

In addressing the *variance problem* for this neural network type, early stopping and ensembling have been shown to be valuable tools. Early stopping sets as small as 5% of the training set have been shown to be effective in reducing the variance error. The current work also suggests that there may be no need for a separate testing set. A validation set of size 45% of the training set can substitute for a testing set 45 times larger, returning an MSE calculation within 7% of the MSE from that testing set $(\sigma = 5\%)$. This offers the possibility of

saving a significant amount of time that would otherwise have been spent sampling for a testing set.

Ensembling has been shown to be more effective than early stopping in reducing variance and, in the limit, will reduce the variance to zero. Novel equations have been presented in this chapter that will provide approximations for the variance and the bias of an ensemble using mean square error calculations alone. The determination of bias and variance contributed by the current work deviates from the theme of metamodelling, and so a comparison with an existent technique fits best in the appendix.

Using the equations given in this chapter, and assuming that ensembling will be applied de-rigueur, if bias is found to be the dominant component of our error then we infer that the information capacity of the CasCor network has been exceeded because the features in the training dataset lie on a highly multimodal response surface. Chapter 4 addresses the bias problem.

# 4 PATCHWORKING AS A TECHNIQUE FOR THE BIAS PROBLEM OF CASCADE CORRELATION NEURAL NETWORKS

## 4.1 Introduction

The three components of a neural network's mapping error are; variance, bias, and noise. Chapter 3 has addressed the variance problem using the known techniques of early stopping and ensembling. When training on deterministic datasets, such as results from CAE solids and fluids solvers, we can expect no experimental noise. The remaining problem is therefore bias, which is addressed in this chapter.

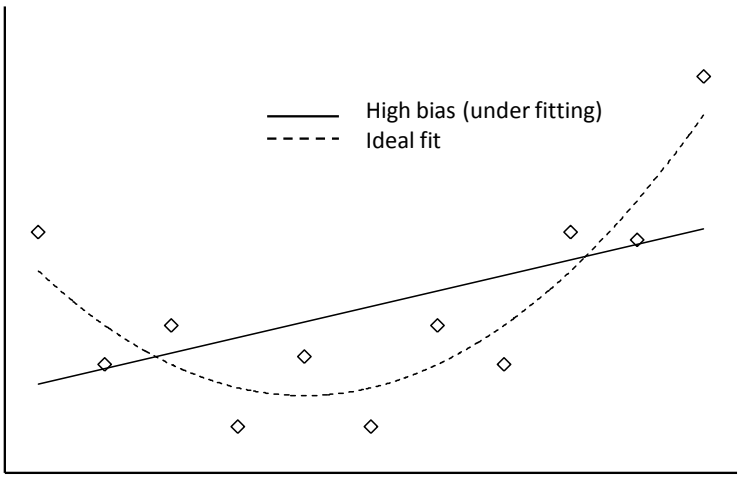



**Figure 4-1 High bias**

Constructive neural networks, such as Cascade Correlation (CasCor) have the potential to solve the bias problem (Figure 4-1) of neural networks by adapting their size to suit the number of features of the problem at-hand. However, as experiments in this chapter will show, the *information capacity* of the CasCor neural network appears to limit its potential. Although not rigorously defined, this *capacity* is a measure of a neural network's ability to represent the features within the training set and roughly corresponds to its ability to model any given

function. It is also related to the amount of information that can be stored in the neural network and to the notion of complexity.

In this chapter, a novel technique, named here as “patchworking”, is introduced that addresses the bias problem of CasCor networks, by raising their information capacity. By using patchworking for domain subdivision the information content in the training sets, and hence the error, is much reduced. The total information capacity of the patchwork has grown – thus improved generalisation on multimodal test functions is obtained. As will be seen, patchworking, when used in combination with early stopping and ensembling, can achieve an order of magnitude improvement in the error.

### 4.1.1 Patchworking - a subdivision method

This technique is particularly suited to highly multimodal response surfaces. Determined empirically, “highly multimodal” is defined as six or more distinct extrema over a multi-dimensional surface; the fit deteriorates significantly when the extrema exceed nine. Functions such as these are used in this chapter to demonstrate CasCor’s difficulty in fitting the underlying function (functions given in the appendix). These poor fits appear as high MSEs on testing sets and are also clearly visible in surface plots. Neither early stopping, nor ensembling, are sufficient to overcome these poor fits as the source of this problem is the inability of a single CasCor neural network to represent the complex features in the dataset.

Some of the greatest strengths attributed to the CasCor type of neural network are as a result of it growing its own topology during training. An intrinsic feature is that at any point during training, no more than one new neuron will be having its weights optimised. It is widely believed that this distinguishing behaviour results in rapid training times; however, this is challenged by [59], in which Squires et al. conclude that freezing of formerly trained weights can be detrimental to effective learning.

The universal function approximation abilities of the CasCor neural network, mathematically proven by Kwok and Yeung [60], are only applicable if we assume that correct choices have been made when each and every neuron was inserted. By taking a system view of the training process, it can be argued that correct choices are frequently not made when mapping multimodal functions.

Informally, the training process plays the role of an agent in the system. This agent aims to train and fix in the network one neuron at a time that, in isolation, reduces the MSE on the training set by the largest possible amount. Several time steps later in the training, more neurons have been added and we see, with the benefit of hindsight, that incorrect choices have been made in the early stages of training. What were once apparently optimal additions to the network are ultimately conspiring to deflect the network from a good mapping of the underlying function. The training algorithm dictates that once neurons have been placed in the network, they may not be removed or re-trained (*weight freezing*) and so the problem becomes irreconcilable [59]. The problem is one of *decision theory* – specifically *evidential decision theory*: how can a training process place a neuron in the network which, later in time, will combine with downstream neurons in only a beneficial way?

A more formal description can be found in [61] where they consider the problems caused when training on the simple “double-tanh” function. The problem is seen to be sufficient to preclude, or at least delay, convergence of the CasCor network. Variants of the CasCor neural network include one that only adds neurons to a single hidden layer (breadth) [47] and one that chooses whether to add depth or breadth to the network [62]. Both have mixed success against the standard CasCor.

In this author’s training experiments with datasets that contain highly multimodal functions (Table A-1), the training problem becomes clearer when monitoring the validation MSE. As the network is training, the insertion of new neurons should be conferring a greater information capacity to the neural network, and the validation MSE should decrease. Inserting the first two or three hidden

neurons does cause a small decrease in the validation MSE, but soon after, this error increases resulting in a very poor generalisation of the underlying function.

The hypothesis behind patchworking is that by subdividing the input domain, the number of extrema that any one neural network must approximate is kept below the multimodal threshold. Hence, CasCor networks with a small number of neurons can approximate the function over each subdivision with a lower MSE. In this way, patchworking overcomes the fundamental *weight freezing problem* of this neural network. Ensembling and early stopping can be used in conjunction with patchworking and are, in fact, logical accompaniments.

### 4.1.2 Patchworking method

The algorithm used to construct the patchwork is shown in Figure 4-3. It allows for a user defined number of subdivisions known as “depth” and can be applied to as many input dimensions as is practical. Note, though, that the number of required networks grows exponentially $2^{(depth \times dimensions)}$ and so this method may not be practical if the dimensions number more than nine or ten. The patchworking technique is shown in Figure 4-2 and is applied as follows:

1. Train at first without subdividing the domain (patchwork depth=0)
2. Test the MSE after this training.
3. Subdivide the input domain if the test error is undesirably high (depth = depth + 1).
4. Create more training samples if necessary and re-train on these subdivisions (or ‘patches’).
5. Repeat steps 2-4 until the testing MSE is satisfactorily low.

A relatively simple algorithm can be constructed to query such a patchwork, assuming that we have stored on file the minimum and maximum bounds of each network’s domain.

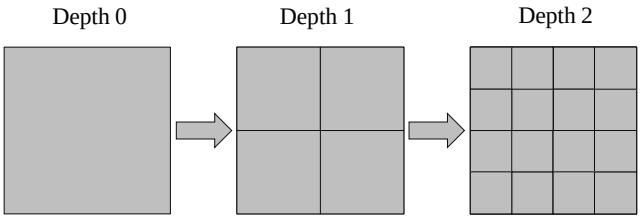



**Figure 4-2 Patchworking subdivisions for a 2D function**

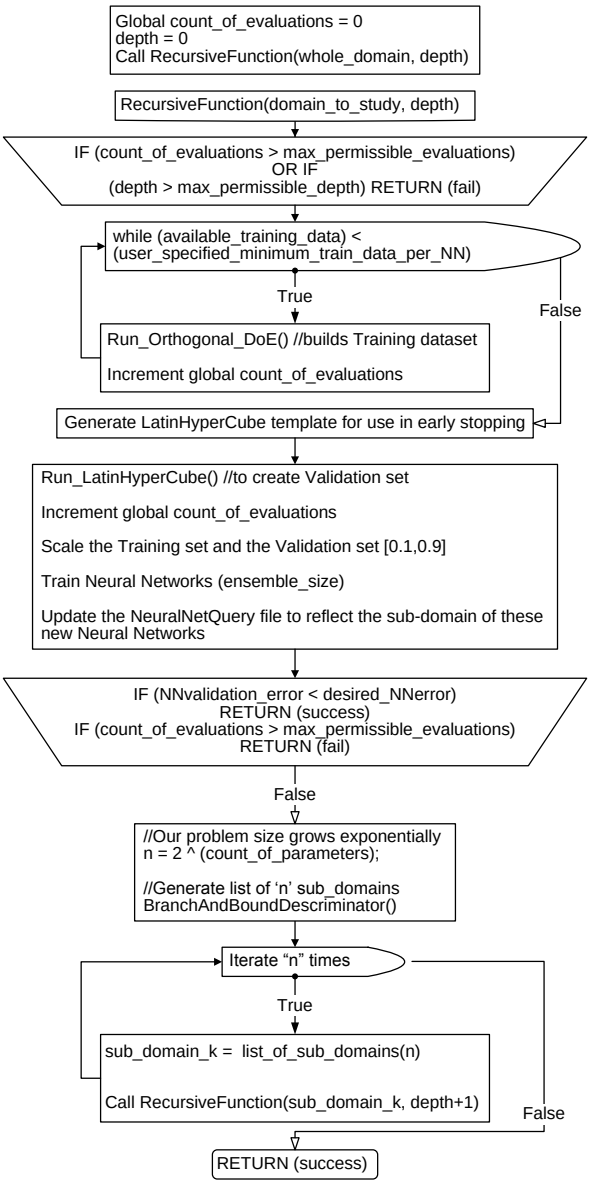



**Figure 4-3 The patchworking algorithm**

Note that there may be surfaces for which patchworking is not an optimal solution: for example, a two dimensional surface where the majority of the features occur in just one quadrant (i.e. one patch). In this example, the remaining three patches are not providing any improvements in the mapping of the three ‘easier’ quadrants, yet they still require more samples upon which to train. A more optimal patchworking solution would apply techniques from analysis of variance (ANOVA) to determine subdivision based upon an evaluation of those sub domains with the highest variance.

## 4.2 Experimental setup

The test functions used are the same thirteen of the previous chapter (Table A-1). Sampling the functions for the training set was likewise performed with an orthogonal array and early stopping and ensembling are applied as described in Chapter 3.

The amount of training data per patch was calculated from the results in Figure 3-3 of Chapter 3 and this is tabulated in column 3 of Table 4-1.

## 4.3 Qualitative results of patchworking (visualisation)

Shown in [Figure 4-4,…,Figure 4-24] are surface plots demonstrating how CasCor fits to the 2D mathematical test functions of Table A-1.

Each set of results is presented as a triplet. The top image is the test function. The middle image is the surface plot of a size 15 ensemble of early stopped CasCor networks. The third image shows clearly the improvement when the patchworking technique has been applied to a depth of 1 along with the techniques of early stopping and ensembling.

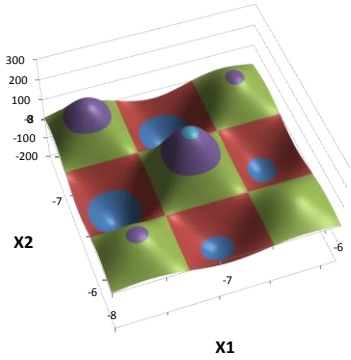



**Figure 4-4 Shubert function**

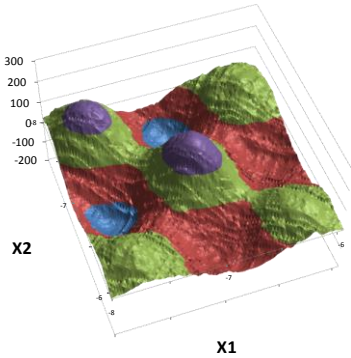



**Figure 4-5 Shubert Ens+ES**

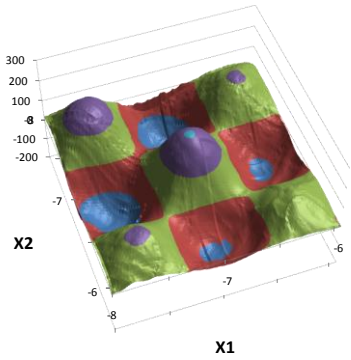



**Figure 4-6 Shubert patchworking+Ens+ES**

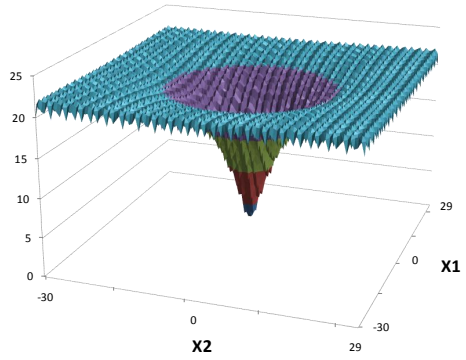



**Figure 4-7 Ackley function**

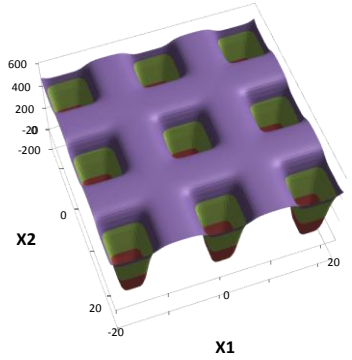



**Figure 4-8 De Jong's 5th function**

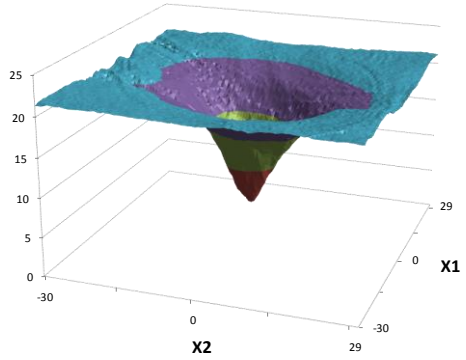



**Figure 4-9 Ackley Ens+ES**

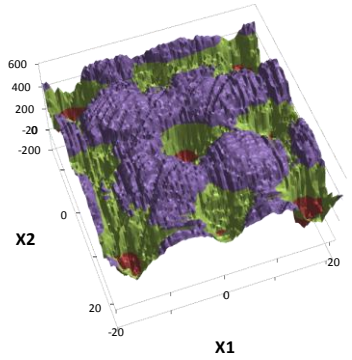



**Figure 4-10 De Jong's 5th Ens+ES**

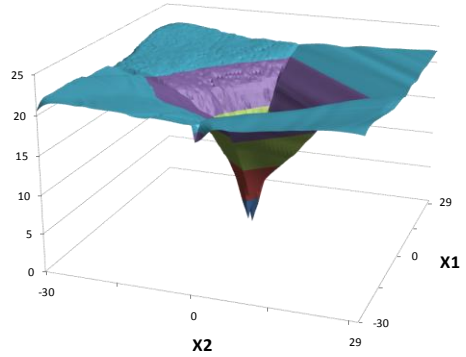



**Figure 4-11 Ackley patchworking+Ens+ES**

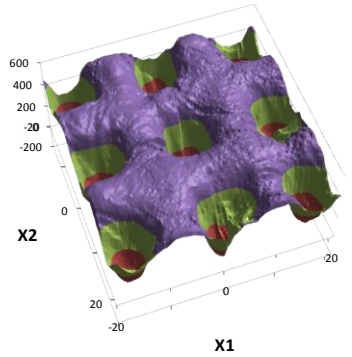



**Figure 4-12 De Jong's 5th patchworking+Ens+ES**

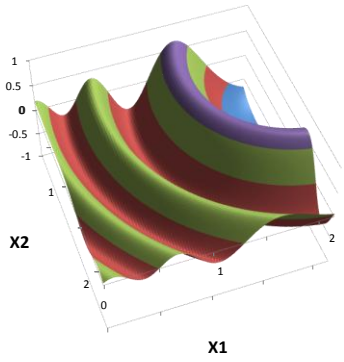



**Figure 4-13 Langermann function**

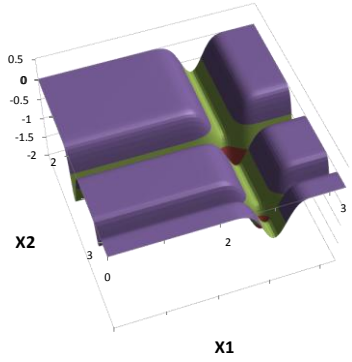



**Figure 4-14 Michalewicz function**

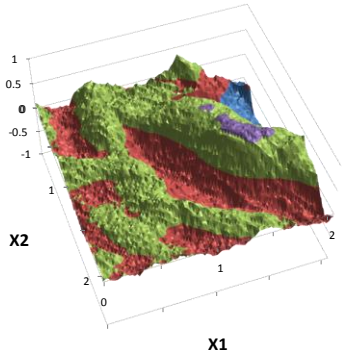



**Figure 4-15 Langermann Ens + ES**

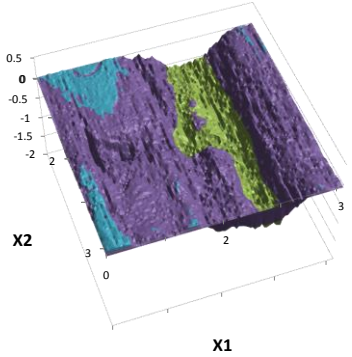



**Figure 4-16 Michalewicz Ens+ES**

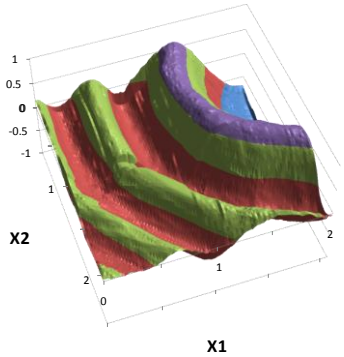



**Figure 4-17 Langermann patchworking+Ens+ES**

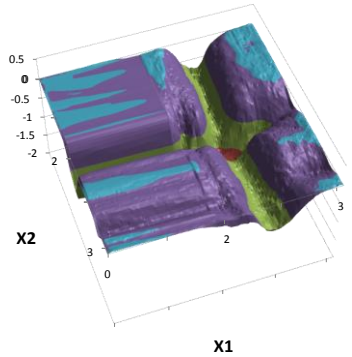



**Figure 4-18 Michalewicz patchworking+Ens+ES**

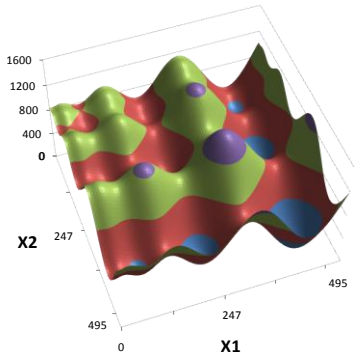



**Figure 4-19 Schwefel function**

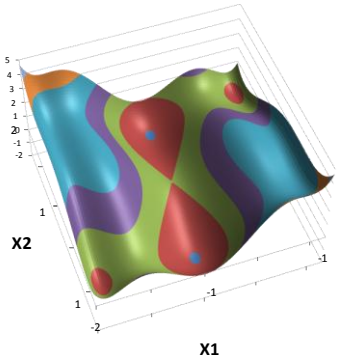



**Figure 4-20 Six Hump Camel Back function**

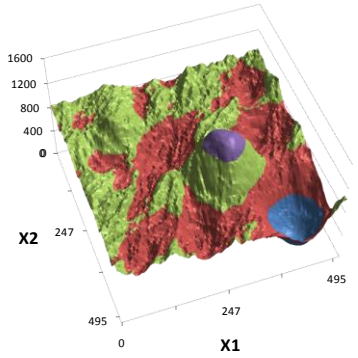



**Figure 4-21 Schwefel Ens+ES**

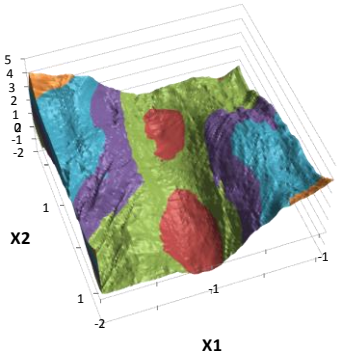



**Figure 4-22 Six Hump Camel Back Ens+ES**

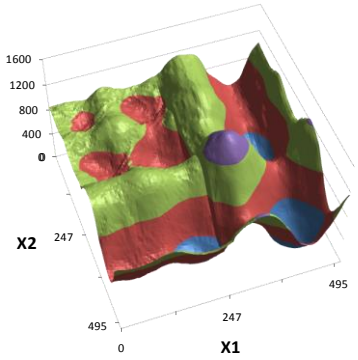



**Figure 4-23 Schwefel patchworking+Ens+ES**

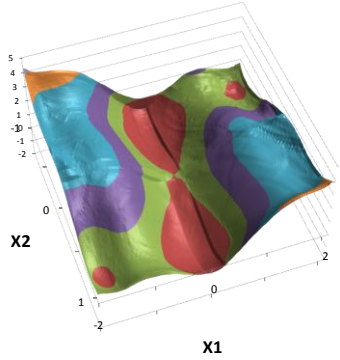



**Figure 4-24 Six Hump Camel Back patchworking+Ens+ES**

## 4.4 Quantitative results of patchworking

In Table 4-1 $\mathrm{Ens}_{\mathrm{size}} = 15$ was used and the basic CasCor results are shown alongside the benefits of ensembling (Ens) + early stopping (ES), patchworking, and all three combined. Patchworking is applied to a depth of one. The same

computer program was used to generate all the neural networks, the only changes being flags that turn on/off the features shown. Results shown are formed from the arithmetic mean of ten trials.

When compared to a standalone CasCor neural network, the mean effect of patchworking is to reduce the error by a factor of 6.2. Employing ensembling and early stopping on these functions reduces the error by a mean factor of 2.8. However, the real benefit of patchworking is that it can be combined with the techniques of early stopping and ensembling – here delivering a mean reduction in neural network testing error of a factor of 11.9 (91.6%).

## 4.5 Patchworking for greater depths and dimensions

From this author's experience with the CasCor neural network, no more than nine features can be mapped satisfactorily by one network alone. Taking the full domain of the two dimensional Schwefel function as an example, Figure 4-25, we see significantly more than nine stationary points on this surface. Patchworking to a depth of one, Figure 4-27, begins to approximate the Schwefel surface but, using the recursive facility of the patchworking algorithm, a significant improvement can be seen in Figure 4-28 when patchworking has been allowed to continue to a depth of three.

## Table 4-1 Patchworking results

| | | | | MSE $\times 10^3$ | | |
|---|---|---|---|---|---|---|
| Test function | Dims | Size of train + early-stop sets. Patchwork Off/On | Cascade Correlation (CasCor) | **CasCor with Ens + ES** | **CasCor with Patchworking** | **CasCor with Patchworking +Ens + ES** |
| Ackley | 2 | 116/461 | 33.79 | 3.10 | 6.31 | 1.53 |
| | | | Reduction in error: | 90.82% | 81.33% | 95.47% |
| DeJongs5th | 2 | 116/461 | 176.33 | 58.10 | 33.20 | 11.23 |
| | | | Reduction in error: | 67.05% | 81.17% | 93.63% |
| Langermann | 2 | 116/461 | 77.33 | 22.43 | 3.82 | 1.48 |
| | | | Reduction in error: | 70.99% | 95.06% | 98.09% |
| Michalewicz | 2 | 116/461 | 22.90 | 10.78 | 5.23 | 3.27 |
| | | | Reduction in error: | 52.92% | 77.16% | 85.72% |
| Schwefel | 2 | 116/461 | 36.73 | 4.39 | 3.77 | 0.80 |
| | | | Reduction in error: | 88.06% | 89.75% | 97.81% |
| Shubert | 2 | 116/461 | 32.08 | 4.59 | 3.11 | 0.27 |
| | | | Reduction in error: | 85.69% | 90.31% | 99.15% |
| Six Hump | 2 | 116/461 | 13.39 | 4.26 | 1.46 | 0.36 |
| | | | Reduction in error: | 68.15% | 89.09% | 97.34% |
| Ackley | 3 | 173/1383 | 14.66 | 5.64 | 4.78 | 2.37 |
| | | | Reduction in error: | 61.56% | 67.38% | 83.84% |
| Hartmann | 3 | 173/1383 | 12.67 | 6.50 | 2.44 | 2.38 |
| | | | Reduction in error: | 48.66% | 80.76% | 81.18% |
| Rosenbrock | 4 | 231/3687 | 18.27 | 8.19 | 4.88 | 2.99 |
| | | | Reduction in error: | 55.18% | 73.27% | 83.61% |
| Schwefel | 4 | 231/3687 | 27.47 | 13.70 | 2.84 | 2.37 |
| | | | Reduction in error: | 50.12% | 89.66% | 91.36% |
| Michalewicz | 5 | 288/9216 | 10.64 | 5.38 | 1.74 | 1.35 |
| | | | Reduction in error: | 49.45% | 83.62% | 87.35% |
| Schwefel | 5 | 288/9216 | 44.77 | 22.07 | 3.66 | 1.55 |
| | | | Reduction in error: | 50.71% | 91.83% | 96.54% |
| **Average reduction in error** | | | | **64.57%** | **83.88%** | **91.62%** |

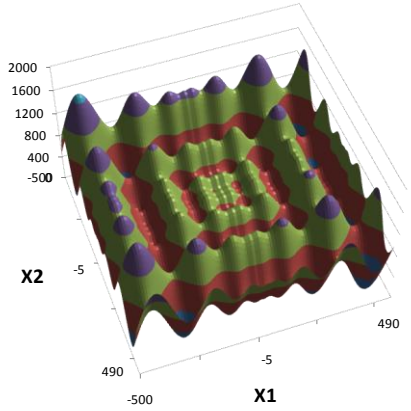



**Figure 4-25 Schwefel function x(i) [-500,500]**

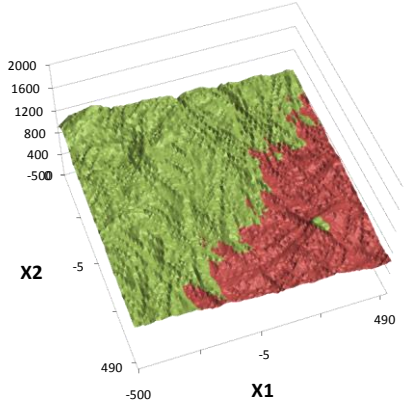



**Figure 4-26 CasCor mapping of full domain of the Schwefel function (Ens + ES)**

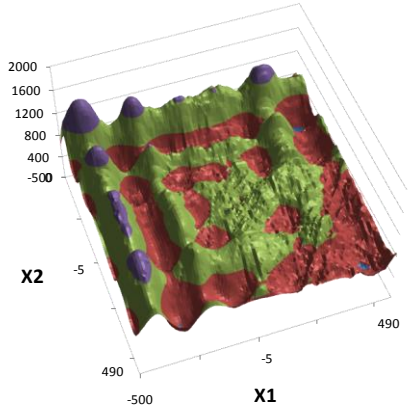



**Figure 4-27 CasCor of Schwefel (Patchworking depth = 1 + Ens + ES)**

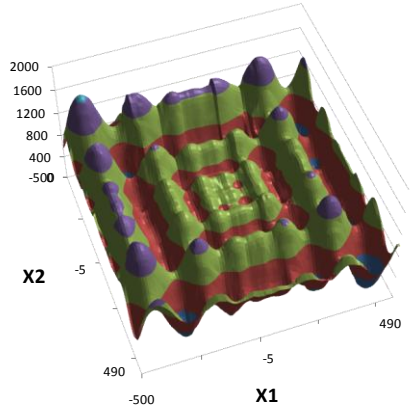



**Figure 4-28 CasCor of Schwefel (Patchworking depth = 3 + Ens + ES)**

The required sizes of training datasets per patch remain the same for any given problem, but the number of patches grows exponentially $= 2^{(depth \times dimensions)}$ therefore, so too will the total training data required. Some fields in which patchworking may be appropriate are those which already have very large datasets e.g. health databases, astronomy data, chemical process data, or any other collection of data samples where the data available is exponentially larger than the dimensions of that data. The information capacity of patchworked CasCor networks also grows exponentially and so it is possible to provide a

useful heuristic rule-of-thumb to calculate the number of features that can be mapped. In the general case:

$$\textit{Maximum features mappable} = 9 \times 2^{(depthxdimensions)} \quad \textbf{(4-1)}$$

Therefore, given an eight-dimensional problem, patchworking to a depth of one could map as many as 2,304 unique features in a training dataset numbering 98,304 samples.

## 4.6 Summary of patchworking

The architecture of the Cascade Correlation neural network means that it is quick and simple to configure for training. However, its *weight freezing* mechanism can introduce undesirably high bias when mapping multimodal functions. Although weight freezing has not been removed by the current author, its detrimental effects can be ameliorated by sub dividing a highly multimodal surface into small domains - each with fewer features.

This introduction of this author's *patchworking* technique reduces the bias component of error by raising exponentially the *information capacity* of the Cascade Correlation neural network. Although patchworking does require exponentially larger training datasets, it overcomes the weight freezing problem of this neural network type and leads to significantly improved fits for multimodal problems - yielding a reduction in error of over ten in some cases.

## 4.7 Usage with real world data

An example of real world usage is the application of the patchworking technique to a dataset that has many thousands of samples; namely census data, specifically house prices in California [63], again from the machine learning repository [1]**.** The original dataset has eight input dimensions and one output dimension with 20,640 samples in the training dataset. The eight inputs are: median house value, median income, housing median age, total rooms, total bedrooms, population, households, latitude, and longitude. The dependant

variable is the median house value. By applying this author's test for the minimum training dataset (patchworking depth=1) it is found that a minimum of 65,536 samples would be needed for eight dimensions, hence there is a requirement to reduce the dimensionality of the input dataset. The following operations were performed; total bedrooms and population is reformed as a ratio; bedrooms/population. Likewise, total rooms and households becomes the ratio rooms/household. With six dimensional data, our test for minimum dataset yields a minimum of 12,288 samples. As there are 20,640 samples in the source data, there now exists sufficient training data to patchwork the census dataset. Of those 20,640 samples, 30% is set aside for our validation / early stopping dataset, leaving 14,447 for the training dataset. Results from a non-patchworked CasCor ensemble are to be compared to a patchworked CasCor ensemble. In both cases, an ensemble size of 15 is chosen, and early stopping is applied using the validation dataset.

### 4.7.1 California house price results

The MSE of the non-patchworked solution was $8.05\times10^{-3}$. For some of the patches in the patchworked solution, specifically those representing houses in regions where the income is in the top 50% and the age of the houses are newer (bottom 50%), the MSEs were similarly high and in all cases represent poor mappings of the underlying function of house value. Speculatively, it could be concluded that these high errors were the result of the response surface being above this author's multimodal threshold of nine features per patch. However, these high errors could also have arisen due to the training dataset having insufficient data to represent all the factors that influence house price. For example; the local geography, the proximity of commercial zones, the proximity of the houses to industrial parks and the transport infrastructure could correlate to house value but are not captured in this census dataset. Nevertheless, the application of patchworking resulted in lower MSEs for several other patches [$3\times10^{-3}$, $4.8\times10^{-3}$]; a reduction of over 50% in the error.

For this real-world dataset, patchworking has successfully raised the overall information capacity, thereby reducing the bias, hence reducing the error.

Table 4-2 shows a small extract of the house price prediction for the patchworked and non-patchworked solution. The mean error in house price prediction of the non-patchworked solution was 20.3%, the patchworked solution's mean error was 11.6%. This translates as predictions of house value (where the mean house price is ~$192,000) having a mean error of $22,350 (patchworked) versus $38,900 (non-patchworked).

**Table 4-2 House price prediction**

| Median income | Housing median age | Bedrooms/pop | Rooms/household | Latitude | Longitude | Real house value (**Expected Prediction**) | Median house value (non-patch) | Median house value (**PATCH**) | Difference (non-patch) | Difference (non-patch %) | Difference (**PATCH**) | Difference (**PATCH** %) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4.31 | 34 | 0.34 | 5.54 | 34.19 | -118.61 | **210100** | 233866 | **220626** | 23766 | 11.3 | **-10526** | **-5** |
| 3.73 | 30 | 0.34 | 5.4 | 35.38 | -118.95 | **83100** | 231720 | **148930** | 148620 | 178 | **-65830** | **-79.2** |
| 4.00 | 30 | 0.25 | 3.69 | 34.28 | -119.16 | **219200** | 103647 | **183066** | -115553 | -52.7 | **36134** | **16.5** |
| 4.19 | 29 | 0.29 | 5.3 | 34.22 | -119.17 | **197100** | 222322 | **212317** | 25222 | 12.8 | **-15217** | **7.7** |
| 3.86 | 30 | 0.31 | 4.79 | 34.21 | -119.18 | **234700** | 242014 | **207547** | 7314 | 3.1 | **27153** | **11.6** |
| 3.93 | 32 | 0.27 | 4.94 | 34.17 | -119.18 | **187600** | 251222 | **203118** | 63622 | 33.9 | **-15518** | **8.3** |
| 4.62 | 32 | 0.24 | 5.03 | 34.18 | -119.19 | **181100** | 228883 | **210826** | 47783 | 26.4 | **-29726** | **-16.4** |
| 4.69 | 35 | 0.32 | 5.23 | 34.39 | -119.3 | **199300** | 236638 | **235460** | 37338 | 18.7 | **36160** | **18.1** |

## 4.8 Limitations of Patchworking

The patchworking method, introduced in this chapter, is a subdivision mechanism for reducing the number of features that any one Cascade

Correlation neural network must map and allows a patchwork of the networks to map highly multimodal functions. Clear improvements generated by this author's patchworking technique have been demonstrated in this chapter; both in the table of results and in the surface plots. The effectiveness of patchworking vindicates the initial hypothesis that this neural network type has an inherent weakness when presented with multimodal functions. Patchworking does offer a workable solution that exponentially raises the capacity of a patchworked Cascade Correlation ensemble. The compromise that we are forced to make, should we need to use patchworking, is the use of training datasets exponentially larger than the dimensions of the problem. If using this neural network type for machine learning, it is conceivable that we already have a plethora of data – the census dataset for example – and we can expect that patchworking will deliver a reduction in the mean squared error by reducing the bias of our patchwork.

It is an unfortunate conclusion that this thirst for training data obviates the use of patchworking to accelerate design optimisation. Even for a three dimensional patchwork, eight patches are required and with $32 \times 3$ training samples per patch the minimum amount of training data would number 768 samples. With reference to the aerofoil optimisation (Chapter 5), a Pareto optimal set of aerofoils was found after only 300 function evaluations. Therefore, if a patchworked CasCor metamodel was applied to accelerate such an aerofoil optimisation, it would in fact more than double the elapsed time compared to not using a metamodel at all.

One further weakness of the patchworking technique is revealed by considering training on data that is greater than three dimensions. To describe this weakness, first consider a three dimensional case. With no patchworking (or patchworking at depth = 0) the CasCor neural network has been found to approximate from one to nine features. Patchworking to depth = 1 yields eight patches. Each of those patches may represent from one to nine features each ($8 <$ total features mappable by the patchwork $< 72$). Hence, a three

dimensional patchwork (depth = 1) continues to map features where a non-patchworked solution leaves off. Now consider a six dimensional case. With no patchworking, once again we can approximate from one to nine features. However, patchworking to depth = 1 yields a patchwork of 64 neural networks (64 < total features mappable by the patchwork < 576). Were the six dimensional problem to contain only 30 features in the training dataset, we would have given our patchwork too much capacity to map these 30 features. Our bias would be very low, but we would have induced the likelihood of high variance. A large size of ensemble would then be necessary to reduce this variance, thus increasing the overall training time.

## 4.9 Conclusion

The principal contributions of this chapter derive from the investigation of functions that cause exceptional mapping problems for the Cascade Correlation neural network. Attempting to train on the full domain of the Schwefel function illustrates a complete failure of this neural network (Figure 4-26). The neural network ensemble's failure is demonstrated by a fall-back to mapping Schwefel's surface with nothing more than a hyperplane. This hyperplane-failure is readily repeatable with any sufficiently multimodal function and is in-no-way unique, conceivably occurring for real world problems as well as for mathematical test functions. Testing the modality of a dataset prior to training may not be possible. Of little consolation is that a complete failure to train on a given dataset is a good indicator of high modality in that dataset i.e. greater than nine stationary points.

The experimental results in this chapter call into question the *universal function approximation capabilities* of the cascade correlation neural network (Kwok and Yeung [60]). In theory, the provision of an unlimited amount of training data, and no cap on the maximum number of neurons, should mean that this neural network type can approximate an arbitrary function. However, this neural network's maximum capacity has been found to be limited to surfaces of minimal complexity; specifically those that can be fully approximated with fewer

than 9 or 10 hidden neurons [64]. One beneficial outcome of this limiting capacity is that we can now calculate the necessary number of training samples prior to training (section 3.3.1). Applying the patchworking technique does exponentially increase capacity - only then is a CasCor patchwork (of unrestricted depth) capable of universal function approximation. Note also that simply removing the weight freezing mechanism is not without complications. Doing so would re-introduce the *moving target problem* of training – a problem circumvented by the original Cascade Correlation neural network.

Only one example was found in the literature of other researchers failing to map multimodal functions with CasCor neural networks, this paper being a theoretical approach using the double-tanh function [61]. No publications were found that highlighted a failure to map real-world datasets. Informally, users of the FANN library report mixed success with CasCor on their forum. Those that initially fail to train on a dataset using CasCor go on to have success with fixed topology neural networks. Are there any approaches other than this author's patchworking technique that may work for multimodal datasets?

A neural network type in the literature, inspired by CasCor, is Constructive Back Propagation (CBP) of Lehtokangas [65]. The "cascading approach" is kept; namely starting with an empty network and letting the topology grow in size, however, two neurons are trained at each time step rather than just one. Lehtokangas does not explore the results for training three, four, or even ten neurons at each time step, although his papers do state that this functionality is supported with his training algorithm [65, 66]. Also, he reduces his error not by correlation but by the back propagation technique. For all benchmarks given by Lehtokangas, CBP is seen to deliver a neural network with lower testing errors than CasCor and without an increase in training time. Unfortunately, further experimentation with CBP has been impeded as the original code has not been made publicly available. CBP can, in principle, be implemented as part of Nissen's FANN library. An attempt was made by the current author to realise such an implementation but it was not possible due to time constraints.

In summary, CBP has the potential to train successfully on the same multimodal surfaces used in this work to illustrate CasCor's weakness. If the CBP training datasets need only grow linearly with the number of dimensions (rather than exponentially for patchworked CasCor networks) then CBP offers an attractive possibility for building a cascading neural network metamodel to assist engineering design optimisation.

# 5 SHAPE OPTIMISATION CASE STUDY

## 5.1 Introduction

This chapter describes the shape optimisations of aerofoils for use with unmanned aerial systems (UAS). As aerofoil optimisation is a long running theme within the research community, the contribution deriving from this chapter's optimisations is minor. The underlying motivation, though, was to use aerofoil optimisation as a case study of a computer-based real-world shape optimisation. The overall aim was to expose the practicalities of such optimisations, with the intent of revealing potential research gaps.

After setting the scene for the UAS scenario (5.1.1), the main body of this chapter first presents a flowchart of a generic optimisation process that is typical for shape optimisation needs (5.1.2). The objectives for this aerofoil optimisation are stated (5.1.3), and a case made for the application of a multi-objective optimisation algorithm (5.1.4). Then described are the software components that were used to create new aerofoils on-demand. Aerofoils with minimal drag and maximal lift were sought; the configuration of the fluid dynamics software that was employed to satisfy these requirements is then described (5.2.4). Results of the aerofoil optimisations are presented in section 5.3, although the discussion that follows in section 5.5 pertains to the overall aim of this chapter. Discussed are; the significant proportion of time in preparing the optimisation jobs and the problems encountered whilst the optimisations were running. Two research gaps are uncovered; since filled by the work of two Cranfield Graduates.

### 5.1.1 The UAS scenario

A dimensionless value often associated with the analysis of the flow of fluids such as air is the *Reynolds number*. For the wings of aircraft this value is proportional to the magnitude of a wing's chord and proportional to its airspeed. Aerofoil optimisation has often focused on finding better aerofoils for manned

aircraft. Having large airframes, and often high cruising speeds, these aerofoils operate at high Reynolds numbers ($>10^6$).

Recent military conflicts have seen a 300% year-on-year surge in the deployment of Unmanned Aerial Systems (UAS) [67]. Classified by either range or size, the close-range or short-range Mini-UAS operate at Reynolds numbers between 75,000 and 150,000. The larger, medium-range, versions typically cruise at greater speeds and have larger chord sizes but, nevertheless, operate at or below Reynolds numbers of $5x10^5$ [68].

In Carmichael's comprehensive NASA report [69] he named 12 distinct regions of interest for Reynolds numbers, ranging from the completely viscous flow at fractional Reynolds numbers to Reynolds numbers as high as $10^9$ in which large nuclear submarines operate. The operating Reynolds numbers for close-range and medium-range UAS place us in two regions of Carmichael's-12; both of which he categorises as operating conditions where extensive laminar flow may be obtained in the boundary layer over much of the surface of an aerofoil. Though he alludes to the desirable effect that this laminar boundary layer can have on reducing the drag of an aerofoil, much attention is also devoted to the highly undesirable effect of the detachment of this layer (a worse effect, seen more frequently for the lower end of the Reynolds numbers under consideration here, is the failure of this layer to re-attach). Detachment of the boundary layer, or the formation of a boundary layer bubble, significantly increases drag and reduces lift and can also initiate a complete stall of an aerofoil at manoeuvring angles of attack (as low as 6 degrees) [70] rather than more usual stalling angles which are typically in excess of 10 degrees.

The idiosyncrasies of each of the 12 bands of Reynolds numbers that Carmichael describes means that aerofoils designed to perform well for large, manned, aircraft cannot be assumed to have optimal performance for the smaller airframes and lower airspeeds of the close-range and short-range UAS - hence the motivation for UAS aerofoil optimisation.

### 5.1.2 The generic form of optimisation

The aerofoil optimisation of this chapter can be readily phrased with respect to the generic design optimisation flowchart in Figure 5-1[71]. The aerofoil template is that of the NACA 4-digit profile, described in section 5.2.2. It has three *design variables* (thickness, camber, position maximum camber). Semantically, a *cost function* pertains to a single objective optimisation such as minimising $(-1 \times Lift/Drag)$. For reasons described in section 5.1.4 we pose this aerofoil problem as a multi-objective optimisation but this represents broadly the same step. Our only constraints here are the domains of our design variables that represent the *search space*.

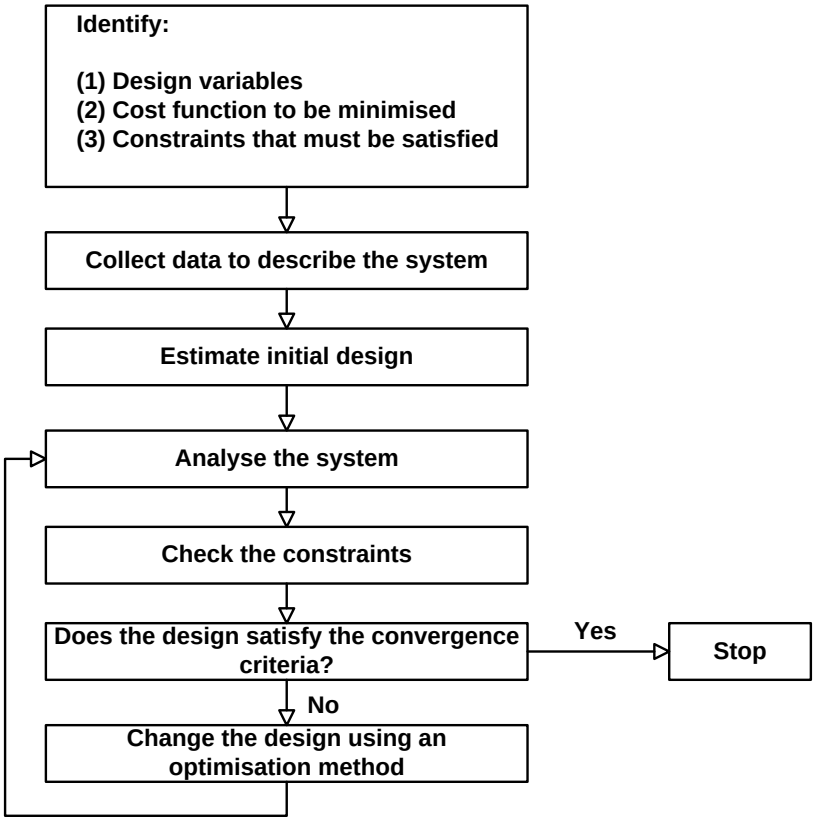



**Figure 5-1 Design optimisation flowchart**

*Collecting the data to describe the system* is analogous to configuring the shape optimisation job, and *estimating the initial design* marks the first step before the

optimisation loop commences; we need an initial design as a benchmark for any permutations of that design to be meaningful.

*Analysing the system* involves the creation of a new aerofoil geometry and the determination of the Lift and the Drag co-efficients of that aerofoil. New geometries are built on-demand as IGES CAD-files by bespoke C++ code written by the current author using the open source CAD Kernel, OpenCascade. These CAD-files are meshed and solved with the use of Computational Fluids Dynamics (CFD) code – in this case, the commercial software of ANSYS (Gambit and Fluent).

*Checking the constraints* are satisfied, and whether *convergence criteria have been met* are functions provided by the chosen optimisation algorithm for this work (section 5.2.1). The demand for new design vectors (i.e. *changes to the design*) is also a product of the optimisation algorithms usage. However, this demand is met by manually coding batch scripts a priori. These scripts link all the parts of the optimisation loop together and enable the optimisation to progress with no user input.

### 5.1.3 Objectives of the UAS aerofoil optimisation

The performance of an aerofoil is often judged by its Lift/Drag (L/D) ratio. Large magnitudes of the L/D ratio, when accompanied by a wide ‘Lift/Drag bucket’, signify optimal designs; having consistently high performance over a wide range of angle of attack (alpha).

The principal objective of this optimisation is to find aerofoils that have optimal values of their Lift/Drag ratio for the Reynolds numbers under consideration.

This aerofoil optimisation could equivalently be phrased as the solution of a single objective (Lift/Drag) or multiple objective (Lift and Drag) problem. A problem shared by either approach is the need to specify the bounds of the search domain. Too large a domain and the optimisation can become very time-

expensive; too small a domain and the discovery of optimal solutions may be obstructed.

The general theme of this thesis is to explore, develop, or enhance methods of reducing the computational load of optimisation. In keeping with that theme, the secondary objective of this chapter is: to reduce the time, and improve the probability of finding optimal aerofoils, for other researchers who are performing similar aerofoil optimisations by determining appropriate domains in which to perform their searches.

### 5.1.4 Applying a multi-objective optimisation algorithm

A hypothesis could be stated that this secondary objective can be found by approaching this aerofoil optimisation with a multi-objective optimiser. With the application of a multi-objective, rather than a single objective, optimisation algorithm we would find multiple optimal aerofoils distributed throughout our search domain. On inspection, the design vectors of these multiple solutions may be found to lie in sub-domains smaller than the initial search space. If the current work can find compact domains (where optimal UAS aerofoils can be expected to be found) then other researchers may benefit by a clear statement of those domains. Any NACA 4-digit aerofoil optimisation that they perform can progress more rapidly in the absence of uncertainty of an appropriate search domain.

For these aerofoil optimisations the multi-objective optimisation, DEMO, is employed. The relevance of DEMO to this thesis is more fully described in Chapter 6. It is sufficient at this point to state that DEMO is a state-of-the-art multi-objective optimisation algorithm.

## 5.2 Experimental setup

### 5.2.1 DEMO

One of the reasons for the popularity of Differential Evolution based optimisers, such as DEMO, is that there are few parameters to tune. The population size is typically chosen as 10 candidates per dimension, although as few as 1.5 candidates per dimension can yield more economic optimisations [72]. The weight can be set between 0.0 and 2.0, and the crossover probability ranges should fall in the domain [0.0, 1.0]. Lower probabilities such as 0.2 and 0.3 are recommended for optimisations with a high number of variables. Higher crossover probabilities are advised for optimisations with only two or three variables [73]. Table 5-1 shows the parameters of DEMO that were chosen.

**Table 5-1 Parameters of the multi-objective optimiser**

| | | | |
|---|---|---|---|
| Population size | 20 | Weight (f) | 0.5 |
| Crossover probability (Cr) | 0.8 | Type of selection procedure | NSGA-II |

### 5.2.2 NACA 4-digit aerofoil type

The NACA 4-digit specification aerofoil is used, in which there are only three optimisation variables; the thickness, the camber, and the position of the maximum camber (expressed as a percentage of the chord where the leading edge is 0% and trailing edge is 100%). The NACA 4-digit specification uses four equations [(5-1),(5-4)] to determine the x and y ordinates for points that would lie on the upper and lower surfaces of the aerofoil. After creating a C++ program, linked to the open source CAD kernel, OpenCascade [74], these points are joined using the GeomAPI_PointsToBSpline method and ultimately output as IGES-type CAD-files for meshing in the commercial software, Gambit (version 2.4).

The NACA 4 digit specification is of the following form $(100\times m)(10\times p)(100\times t)$ such that the NACA 4310 aerofoil has a maximum camber, $m$ of 4%. The position, $p$, of the maximum camber occurs 30% of the chord length from the leading edge. The '10' specifies that this aerofoil would have a maximum thickness of 10% of the chord length. In order to plot an NACA 4-digit aerofoil, the following equations are use to calculate the x and y ordinates of the points on the upper and lower surface of the aerofoil:

$$x_U = x - y_t \sin\theta \quad \textbf{(5-1)}$$

$$y_U = y_c + y_t \cos\theta \quad \textbf{(5-2)}$$

And

$$x_L = x + y_t \sin\theta \quad \textbf{(5-3)}$$

$$y_L = y_c - y_t \cos\theta \quad \textbf{(5-4)}$$

There are two equations for $y_c$ and $\theta$ for ordinates fore, and aft, of the position of maximum camber:

when $0 < x \le pc$ **(5-5)**

$$y_c = \frac{mx}{p^2}(2p - \frac{x}{c}) \text{ and } \theta = \arctan[\frac{2m}{p^2}(p - \frac{x}{c})]$$

when $pc < x \le c$ **(5-6)**

$$y_c = m\frac{c-x}{(1-p)^2}(1 + \frac{x}{c} - 2p) \text{ and } \theta = \arctan[\frac{m}{(1-p)^2}(\frac{c-2x}{c} + 2p - 1)]$$

where

$p$ is the position of the point of maximum camber as tenths of the chord

$m$ is the camber of the aerofoil as hundredths of the chord

$c$ is the length of the chord

$x$ is the position from $0$ to $c$ (for this work, $x$ is incremented from $0$ to $c$ in steps of $c/100$)

Lastly, the thickness distribution $y_t$ is calculated from the following polynomial:

$$y_t = 5tc[0.2969 \times \sqrt{\left(\frac{x}{c}\right)} - 0.1260\left(\frac{x}{c}\right) - 0.3516\left(\frac{x}{c}\right)^2 + 0.2843\left(\frac{x}{c}\right)^3 - 0.1015\left(\frac{x}{c}\right)^4] \quad \textbf{(5-7)}$$

where

$t$ is the thickness of the aerofoil as a percentage of the chord.

### 5.2.3 Aerofoil objective functions

The objective functions that are returned to DEMO require some care in their formulation. A mistake to be avoided is to fix the angle of attack at an arbitrary value, or, to let the angle of attack be an optimisation variable. Undesirably thin aerofoils will be obtained from such an optimisation (Figure 5-2). After testing the performance of such a thin profile, it was found to have a narrow *lift/drag bucket*. Conceivably, it would also have poor structural rigidity and it is likely that it would be subject to the undesirable property of *flutter*.

Proposed is the following simple but effective mechanism to preclude the discovery of such thin solutions; every newly generated aerofoil will be evaluated in Fluent for its co-efficient of lift ($c_l$) and its co-efficient of drag ($c_d$) at two distinct angles of attack. An alpha of one degree and an alpha of four degrees is used. The mean of these two $c_l$ s and two $c_d$ s are then passed to DEMO as the two objective function values. In this way, optimisations will be seen to diverge from seeking inappropriately thin aerofoils because solutions with lift/drag buckets narrower than three alpha will be penalised.

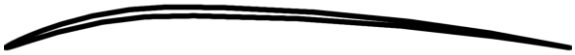


**Figure 5-2 Inappropriately phrased optimisations can converge to give very thin and highly cambered solutions**

### 5.2.4 CFD solver setup and search domain

The Fluent CFD software from ANSYS is used to calculate co-efficients of lift and drag for Re. 75,000 and 250,000. These two Reynolds numbers were chosen as they are representative of small and medium sized UAS and also because of the availability of wind tunnel tests [70] against which the CFD models can be validated.

In the preliminary stages of modelling the lift and drag co-efficients of these aerofoils, several different turbulence models were tested in the CFD software and the resulting lift and drag values were compared to available experimental data. The objective was to determine the most appropriate turbulence models. The Spalart-Allmaras (SA) 1-equation turbulence model [75] (widely used in external airflow simulations) was first tested, followed by 2-equation turbulence models such as the k-epsilon. Lastly, the 5-equation Reynolds-stress model was tested. This latter model was ultimately not selected as it is much more computationally (and therefore time) demanding, hence unsuitable for an optimisation involving hundreds of CFD evaluations.

The two-equation turbulence models were found adequate for the present study. The standard k-epsilon is adapted to free-shear layer flows with relatively small pressure gradients [76]. It is widely used in turbulent flow applications because of its general applicability, robustness and economy [77]. However, this model performs poorly when separation occurs: separation is often under-predicted and/or is predicted too late. A reduced separation usually results in an optimistic prediction of machine performance which could have

dangerous consequences - for example an inaccurate evaluation of when an aerofoil stalls. Hence the standard k-epsilon model was not selected. To avoid further inaccuracies associated with aerofoils operating near the stall, the current airfoil optimisation is limited to airflow conditions and incidences that do not lead to stall events (i.e. values of alpha less than 6 degrees). Note, however, that the two turbulence models found adequate for this work typically perform better than the standard k-epsilon model in predicting near stall phenomena.

For the Re=75,000 case, the Renormalization Group two-equation k-epsilon turbulence model (RNG k-epsilon) was used. This employs a scheme to consider the near-wall flow effects [78]. The enhanced wall-treatment method was selected, which allowed a coarser mesh than that of a low Reynolds k-epsilon model in the viscosity-affected near-wall region with little impact on the accuracy of the simulation in that region. This model is robust in situations with stagnation and separation.

For the second optimisation (the Re=250,000 case), shear stress transport SST [79] uses the simple and robust near-wall formulation of the k-ω model, and switches to the k-epsilon turbulence model in the bulk flow. This model has proved to give accurate results for a wide range of grid densities [80] and avoids the deficiencies of the k-epsilon model (over-prediction of the turbulence length scale – resulting in an over-prediction of the heat transfer at reattachment).

For the Re=75,000 case, the wall treatment and resolution are as follows: first boundary layer row of 0.0002 growing by 20% for each row to a total of 10 rows. Settings are the same for the Re=250,000 case except that the first row begins at 0.0001.

Table 5-2 gives the details of the settings used for the Fluent and Gambit jobs. In Figure 5-3 and Figure 5-4 screen shots of the mesh are displayed. The larger region is a structured mesh; in the proximity of the aerofoil an unstructured

mesh was used as this was found to adapt well to a large variety of aerofoil geometries.

**Table 5-2 Setting for the shape optimisation**

| | |
|---|---|
| Mesh | ~ 120,000 mesh elements |
| Solving algorithm | RNG k-epsilon (Re=75,000)<br>SST: k-$\omega$ near wall, k-epsilon in bulk flow (Re=250,000) |
| Fluent / Gambit version numbers | 6.3 / 2.4 |
| Steady state solution for Cl and Cd | Air considered as incompressible |
| Pressure ambient | 100.920 kPa |
| Gravity | 9.81m/s^2 |
| Density (rho) | 1.2041 (kg/m^3) |
| Velocity for Re = 75,000 Re = 250,000 | 2.193 m/s , 7.3 m/s |
| Chord | 0.5 m |
| DEMO Lift and Drag objective functions formed from the mean of Cl and the mean of Cd at alphas: | 1 degree and 4 degree |
| DEMO Search domain (thickness) | [3,12] |
| DEMO Search domain (camber) | [1,9] |
| DEMO Search domain (position max camber) | [25,52] |

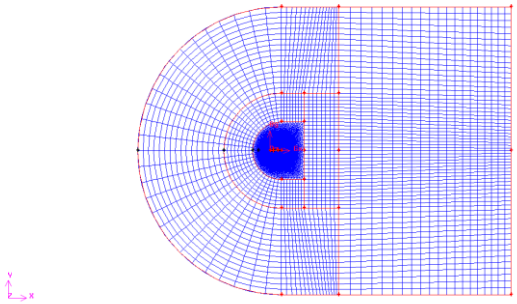


**Figure 5-3 Gambit mesh**

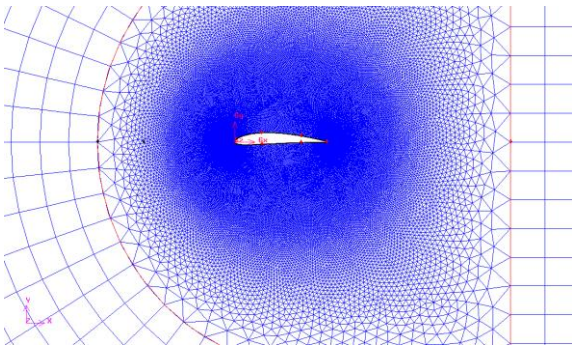


**Figure 5-4 Zoom of Fig 3 showing the unstructured part of the mesh nearest the aerofoil**

### 5.2.5 Validation

The reference aerofoil that is used to validate the CFD model was the NACA 4412 (Figure 5-5). This is an aerofoil design that is still in use today and can be found on aircraft such as the Barrows Bearhawk, the Aeropro Eurofox and the Ayres SR2 [81]. The validation data comes from wind tunnel tests performed by Lnenicka & Horeni in 1978. The data from the original hand-drawn graphs have been re-plotted in Microsoft Excel for clarity. Shown in Figure 5-6, Figure 5-7, Figure 5-8 and Figure 5-9 are the wind tunnel results for Re 75k and 250k alongside the CFD results for the same airfoil at these Reynolds numbers. For solutions at angles of attack where $\alpha > 5^{\circ}$ the results reduced in accuracy when compared to the wind tunnel data, and the CFD solutions were unable to converge satisfactorily as stall was approached. However, aerofoil optimality is rarely based on high values of L/D for a stalled wing. The CFD results for the angles of attack under consideration ($\alpha = 1$ and 4 degrees) match closely the wind tunnel data for the NACA 4412.

**Figure 5-5 NACA 4412**

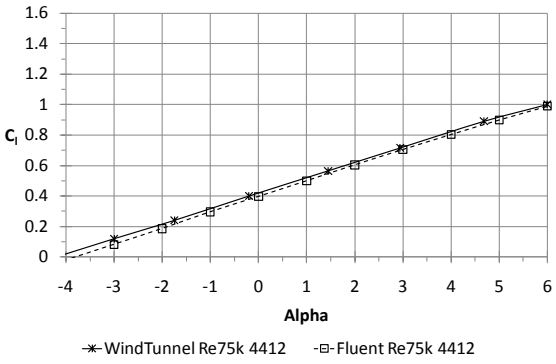



**Figure 5-6 Co-efficient of lift validation Re=75k**

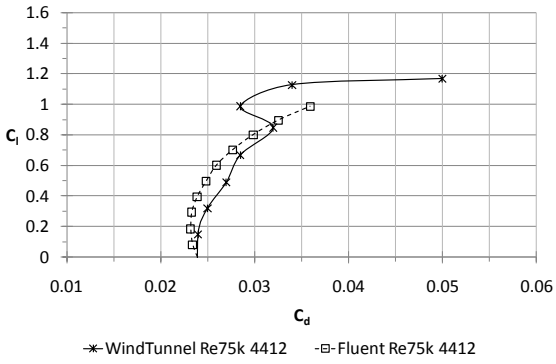



**Figure 5-7 Lift / drag polar validation Re=75k**

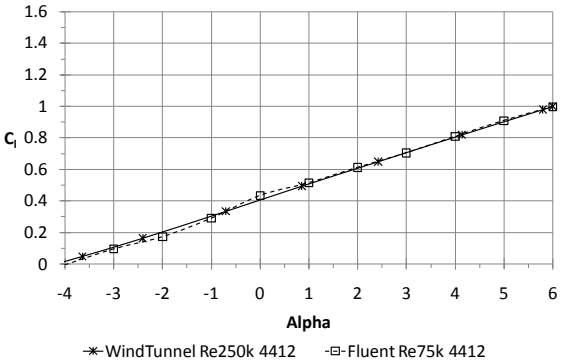



**Figure 5-8 Co-efficient of lift validation Re=250k**

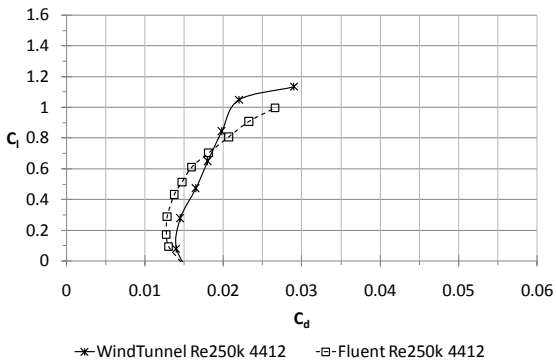



**Figure 5-9 Lift / drag polar validation Re=250k**

## 5.3 Results

For both DEMO optimisations, the Pareto set became fully populated with 20 candidates after four generations. Both optimisations were halted after 15

generations. DEMO is designed to maintain a distributed set of Pareto optimal solutions hence many of these candidates had very low co-efficients of drag (L/D between 5 and 25). However, we are principally concerned with the high L/D solutions and the domains in which they were discovered. Shown in Figure 5-10 and Figure 5-11 are two optimal aerofoils for Re=75,000. They are named Candidate A and Candidate B. For Re=250,000, the optimal aerofoils in Figure 5-14 and Figure 5-15 are named Candidate C and Candidate D.

### 5.3.1 Reynolds 75,000

**Figure 5-10 Candidate A t=3.06 c=5.68 p=35.9**

**Figure 5-11 Candidate B t=5.75 c=8.78 p=50.1**

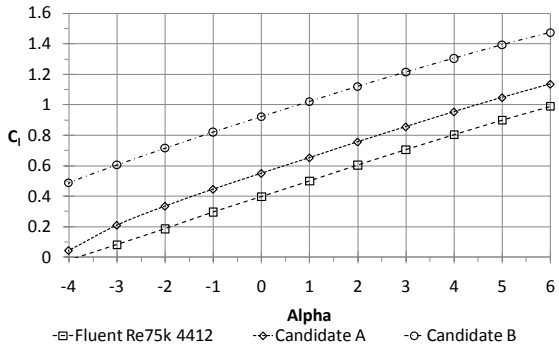



**Figure 5-12 Co-efficient lift plot Re=75k**

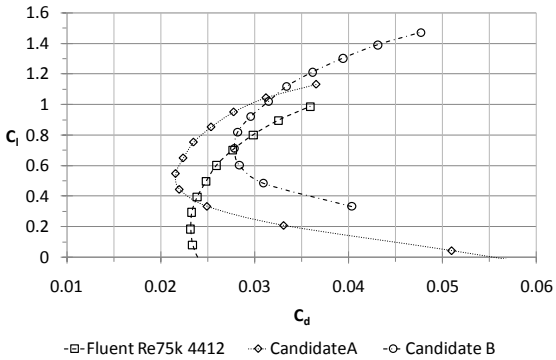



**Figure 5-13 Lift drag polar Re=75k**

**Table 5-3 Lift/Drag for Re=75,000**

| Alpha | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | Peak L/D | Alpha for peak L/D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4412 L/D | -0.8 | 3.6 | 8.1 | 12.7 | 16.6 | 20.1 | 23.3 | 25.5 | 26.9 | 27.6 | 27.5 | 27.6 | 5 |
| Cand A L/D | 0.9 | 6.3 | 13.4 | 20.3 | 25.5 | 29.1 | 32.2 | 33.7 | 34.3 | 33.6 | 31.1 | 33.7 | 3 |
| Cand B L/D | 15.7 | 21.3 | 25.6 | 29.1 | 31.2 | 32.5 | 33.5 | 33.6 | 33.1 | 32.3 | 30.9 | 33.6 | 3 |

Candidate A is the thinner of the two solutions. Its reduced camber delivers a lower $c_l$ than Candidate B, but associated with this is significantly reduced drag (Figure 5-13). The reference airfoil, NACA 4412, has a maximum L/D of 27.6 at alpha=5. Candidate A exceeds this value for all alpha [1,6]; by 17% on average and by 22% at alpha=3 (Table 5-3). Candidate B also exceeds the best L/D of the NACA 4412 for all alpha [-1,6] (16% on average, 22% at alpha=3).

### 5.3.2 Reynolds 250,000

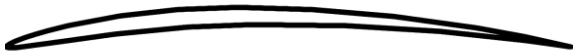


**Figure 5-14 Candidate C t=3.22 c=5.45 p=42.9**

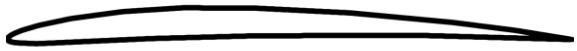


**Figure 5-15 Candidate D t=5.21 c=3.09 p=39.1**

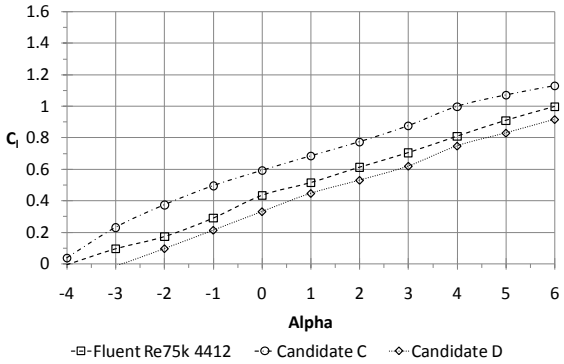



**Figure 5-16 Co-efficient lift plot Re=250k**

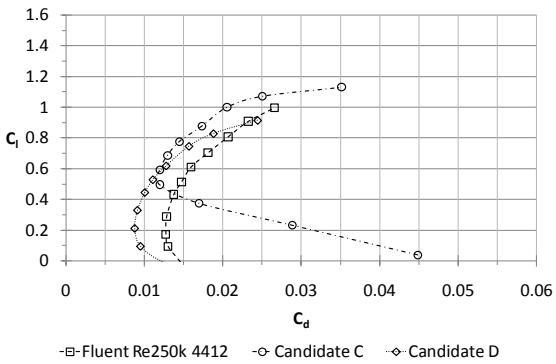



**Figure 5-17 Lift drag polar Re=250k**

**Table 5-4 Lift/Drag for Re=250,000**

| Alpha | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | Peak L/D | Alpha for peak L/D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4412 L/D | -0.3 | 7.4 | 13.6 | 22.6 | 31.6 | 35.0 | 38.4 | 38.9 | 39.1 | 39.1 | 37.6 | **39.1** | **5** |
| Cand C L/D | 0.9 | 8.0 | 22.1 | 41.5 | 49.6 | 53.0 | 53.7 | 50.6 | 48.7 | 42.8 | 32.2 | **53.7** | **2** |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cand D L/D | -4.9 | -1.0 | 10.1 | 24.3 | 36.3 | 44.4 | 47.8 | 48.5 | 47.6 | 44.1 | 37.5 | **48.5** | **3** |

For the NACA4412 reference aerofoil, the mean L/D for alpha [-4,6] has increased from 17.4 at Re=75,000 to 27.5 for Re=250,000. Likewise, the optimal aerofoils found by the DEMO optimisation have increased L/D values. Candidate C exceeds the peak L/D of this NACA 4412 for all alpha [-1,5] by a mean of 24%. Candidate D has a lower camber than both the NACA 4412 and candidate C, giving reduced co-efficients of both lift and drag. Nevertheless, for alpha [1,5], it has a maximum L/D that is 24% greater than the maximum of the NACA 4412 (19% mean improvement).

### 5.3.3 Optimal search domains

Based on this author's inspection of the Pareto optimal sets, it is now possible to state four appropriate domains where other researchers may discover optimal NACA 4-digit aerofoils. The L/D values achievable are similar in each domain, however the domains are classified as either "high co-efficient of lift" or "low co-efficient of drag" to indicate the dominant feature of that domain.

| Domain contains | Thickness | Camber | Position max camber |
|---|---|---|---|
| Re75k High lift | [3,6] | [7,9] | [38,51] |
| Re75k Low drag | [3,4] | [4,6] | [35,45] |
| Re250k High lift | [3,6] | [5.4,8] | [40,50] |
| Re250k Low drag | [4.7,6] | [2,4] | [39,50] |

## 5.4 Aerofoil optimisation - summary

Four optimal aerofoils have been found by this optimisation; two for short range UAS and two for medium range UAS. For each UAS type, the pair of aerofoils found had similar L/D values; both were much improved over the NACA 4412 reference aerofoil.

Each pair of optimal aerofoils found has different properties; one offers a design that could mean reduced cruising speeds but shorter take-off runs (the high-lift candidate), whereas the lower drag candidate would likely require a longer take-off run but offers the possibility of realising a UAS that would have a higher cruising speed.

The secondary objective for this UAS aerofoil optimisation was to determine more compact search domains for NACA 4-digit aerofoil optimisation at these Reynolds numbers. Inspection of the location of the Pareto candidates has enabled these smaller domains to be determined and it is hoped that other researchers may benefit from this minor contribution.

## 5.5 Discussion

The time taken to generate each objective function consisted of two CFD evaluations (at alpha 1 and alpha 4), and was approximately 35 minutes and 70 minutes for the Reynolds numbers of 75,000 and 250,000 respectively. With 300 evaluations as the stopping criteria, and discounting the problems discussed below, the first optimisation run took one week; the other, two weeks. Do these times comprise the greatest cost of this optimisation? The answer is no.

It was decided at the outset of this work to host the meshing of Gambit and the CFD evaluations of Fluent on Cranfield's supercomputer. The optimisation algorithm itself ran on a separate, but networked, Linux PC. As mentioned in section 5.1.2, the co-ordination of the optimisation job was controlled by batch files - specifically, *bash expect* scripts. Logging in and out of this author's supercomputer account, job-submission/control and file transfer were hand scripted in this way. Configuring the bash scripts themselves required two weeks of work.

Writing the C++ OpenCascade code that generated the aerofoil geometries was also a task needing two weeks of coding. However, this statement of time is

misleading; a one month training course on the OpenCascade CAD kernel was first necessary to develop familiarity with its library of functions. We can say at this point that the three weeks of actual aerofoil optimisation comprised only 27% of the elapsed time for the work in the chapter.

Significant problems occurred during the optimisation runs themselves. The following list is not complete, but details some of the problems:

- Cache battery failure on the supercomputer – required a shutdown of the supercomputer (optimisation lost; needed restart)
- Overloaded nodes on the super computer – required a restart of the supercomputer (optimisation progress lost; needed restart)
- No nodes available for use (optimisation paused for up to four hours on each instance)
- Power cuts (optimisation progress lost; needed restart)
- Lack of Fluent licence files – all licences used by other students/academics (optimisation paused for up to four hours on each instance)
- Meshing failure of Gambit (scripts returned an incorrect objective function, corrupting the Pareto set)

These problems shed light on many of the practical problems for any researcher wishing to conduct an automated computer-based shape optimisation and suggest research gaps worthy of investigation.

The first research gap derives from the observation that the numbers of licences for the use of commercial software is typically a finite number. Optimisation is an inherently parallelisable process; for this work the 20 aerofoil candidates of each population can, in theory, be evaluated concurrently on a Grid or cluster of computers. This could yield up to a 20-fold reduction in optimisation time (three weeks becomes 25 hours). However, insufficient availability of commercial licences would preclude this concurrency.

The concurrency limit imposed by employing commercial fluids solvers does not exist if we were to approach this aerofoil optimisation with the use of an open source fluids solver such as OpenFoam. Using the OpenCascade aerofoil generation software of this chapter, Chapman implemented a proof-of-concept

aerofoil optimisation using CAE Linux and OpenFoam [82] in 2009. Although not parallelised for the work in his thesis, the lack of a licensing constraint opens the door to dramatically accelerating shape optimisations when based around open source fluids and solids solvers. In addition, Chapman's experiments with using Python scripts (to automate his aerofoil meshing) comprised an essential component of the work of the current author in Chapter 6. Rather than hand coding the CAD model of Chapter 6's wall-bracket in C++, python scripting reduced the required model set-up time to just a few days of work.

In 2009, Debreuil [83], approached several of the other problems bulleted above. Most of these problems highlighted a need for formalised job submission, job control, and error trapping/reporting. Bash scripting has too little flexibility for this purpose. The management of a shape optimisation that requires access to multiple compute resources is best achieved through workflow management software.

Addendum: Since Debreuil's and Chapman's work in 2009, the Monash Nimrod team have released Nimrod/K (work flow management software) which addresses the type of problems discussed above [84].

# 6 ENABLING MULTI-OBJECTIVE OPTIMISATION IN NIMROD/O

## 6.1 Introduction

The Linux-based Nimrod toolkit provides many useful features: design of experiments, grid/cloud/cluster interfaces for concurrent design evaluations, and an optimisation module, Nimrod/O. Multi-objective optimisations were previously possible only by phrasing them as a single objective optimisation with the use of a penalty function. The contribution of this chapter concerns the interfacing of a truly multi-objective optimisation algorithm. The concurrency possible with Nimrod/O is exploited by introducing a *BatchSize* parameter to the multi-objective optimiser. This forms a secondary contribution of this chapter.

The layout of this chapter is as follows: the Nimrod/O optimisation tool and the chosen optimisation algorithm, DEMO, are described in more detail in sections 6.2.1 and 6.2.2. Adaptations to DEMO that enable concurrency, and the role of the DEMOinterface are detailed in section 6.2.3 before the two test experiments are presented (6.3 and 6.5). The first test is a two-parameter optimisation of a mathematical test function. The second test is the shape optimisation of a rib-reinforced steel bracket using Finite Element evaluations from Code_Aster to compute the two objective functions of stress and deflection as well as incorporating a third, conflicting objective function, of reducing the mass of the part.

## 6.2 Software components

### 6.2.1 Nimrod/O

Nimrod/O combines optimisation, distributed computing and rapid prototyping in one tool. Various optimisation routines are built into Nimrod/O such as BFGS (Broyden–Fletcher–Goldfarb–Shanno), the Downhill Simplex Method, Simulated Annealing, and EPSOC (Evolutionary Programming using Self-Organised

Criticality) [85]. An optimisation is readily specified by the user by parameterising their problem using Nimrod/O's declarative "plan file" (Figure 6-3), after which the tool computes the parameters that minimise or maximise the design's objective function. Transparent to any of the optimisation algorithms is Nimrod/O's evaluation of the objective function. Multiple objective functions can be concurrently evaluated; on a multi-core CPU on the local machine, or by farming out this work to greater compute resources such as a cluster (e.g. [86]), or a grid resource such as provided by Nimrod/G [87], as shown in Figure 6-1.

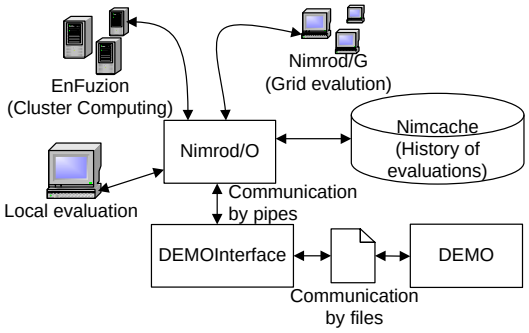



**Figure 6-1 Overview of the process**

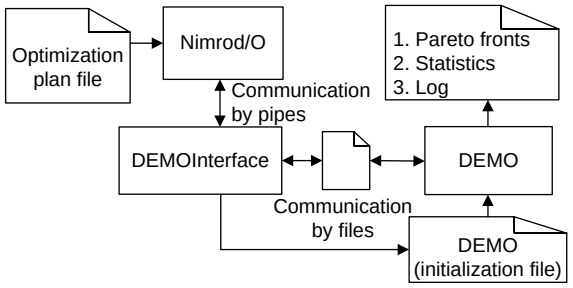



**Figure 6-2 Dataflow between the software elements**

### 6.2.2 DEMO

Differential evolution (DE) by Price [88] was the culmination of work aimed at solving the Tchebychev polynomial fitting problem proposed to him by Dr R Storn. It is a population-based optimisation algorithm, but unlike classical genetic algorithms such as Holland's [89], which bit-encodes decision variables, DE uses floating point encoding. This, coupled with Price's desire to make candidate mutation an adaptive procedure, resulted in a rapid and robust algorithm that is simple to use. The original version of DE is controlled by just three variables: the population size, N, the mutation scaling Factor, F, and the crossover constant, CR.

In creating DEMO (Differential Evolution for Multi-objective Optimisation), the authors, Robič and Filipič , addressed the two goals of multi-objective optimisation [44]:

1. Finding the most diverse range of these solutions across the Pareto set

2. Discovering solutions as close as possible to the true Pareto front.

Based on DE, DEMO builds on the success of Price's algorithm and adds the mechanisms of non-dominated-sorting and crowding-distance-metric as used by other state-of-the-art multi-objective evolutionary algorithms. Tight clusters of non-dominated solutions limit the diversity of the elements in a Pareto set. Penalising this behaviour with DEMO's crowding distance metric helps to achieve the first goal of finding the most diverse range of solutions. The second goal is achieved by an emphasis on elitism: parent individuals are immediately replaced by the candidate that dominates them. By entering the population immediately, this new candidate can, without waiting for the next generation, take part in the creation of further candidates. With these additions, DEMO is shown to achieve competitive results on five ZDT [90] test problems. In a follow-up paper, Robic [91] presents a comparison study in which DEMOS's performance is found to be comparable to other state-of-the-art multi-objective evolutionary algorithms on nine newer test problems created by Huband et al. [92].

### 6.2.3 Interfacing DEMO with Nimrod/O

The original DEMO code was first ported from its Microsoft Windows source code so that it could be compiled under the Linux operating system. The random number generator, a container declaration and the system-out calls comprised the three necessary alterations. Initial testing confirmed that the Linux port of DEMO worked equivalently to the Windows version.

As described in section 6.2.2, one of DEMO's key mechanisms is elitism within the reproduction process. Before an entire population has been evaluated,

superior candidates will already have replaced their parents and taken part in the creation of newer candidates. It should be clear that this mechanism requires sequential candidate evaluation and presents a conflict of interest. Whilst this elitism mechanism should accelerate the discovery of the Pareto optimal set, concurrent evaluations of multiple candidates would reduce the wall-clock time for optimisation runs. To this end, the current author has introduced a *BatchSize* parameter to DEMO's initialisation file.

$$\mathrm{BatchSize} \leq \mathrm{P} \leq \mathrm{N} \qquad \textbf{(6-1)}$$

where

N = Population size, and

P= Number of machines available for concurrent objective function evaluations.

In the case that the user has access to a large computing resource, the BatchSize parameter tunes-down the benefit from elitism in favour of the overall speed-up gained by concurrent evaluations of an entire batch of candidates. Modifying the DEMO source code to enable concurrent candidate evaluations is a contribution of the current work.

An important problem that should be noted is that, after experimenting with enabling or disabling elitism in the Linux version, convergence to the Pareto front did not seem to alter. Furthermore, enabling elitism for the original Microsoft Executable occasionally caused the DEMO executable to freeze after the first population of candidates had been retuned to DEMO. The reason for this freeze is unknown, but may relate to the 64-bit Microsoft Windows operating system on which DEMO was tested (Robič and Filipič's executable was compiled using Borland C++ for 32-bit Windows systems). Further work is planned that will verify the functionality of elitism in the Linux port.

One minor change to DEMO's initialisation file is the inclusion of a Boolean flag that indicates to DEMO that it will be working in a mode compatible with

Nimrod/O. If this flag is turned off (0), then DEMO will function in stand-alone mode and identical to version 1.2. More information on DEMO's usage can be found in the v.1.2 reference manual [73].

A further contribution of the current work is the modifications to the Nimrod/O 2.9 source code that enable an external, multi-objective optimiser, to communicate with Nimrod/O (and also access `stdin` and `stdout`) without *cross-talk*. Via a "`results`" parameter in the plan file (Figure 6-3), Nimrod/O prepares to accept multiple objective functions and, during run time, both logs and caches these multiple results. As in prior versions of Nimrod, the cache mechanism (Figure 6-1) prevents unnecessary repetitions of prior function evaluations. The management of Pareto optimal sets, Pareto based ranking and sorting is not supported by the current version of Nimrod/O (v2.9), however DEMO provides this functionality.

Nimrod/O can host a concurrent execution thread in which an external optimiser runs. This intent is communicated in the plan file by the use of "`method external "name" ./executable`". For the current work, the *pipes* method was chosen. In building the interface, the necessary *include files* from Nimrod/O's package were `noclient.c`, `noclient.h` and `definitions.h`. These provide query and communication functionality between external code, such as the current interface, and Nimrod/O. Sufficient functions are implemented in `noclient.c` that an external, user-defined, optimisation algorithm can operate as if it were part of Nimrod/O.

The DEMOinterface is simultaneously the child process of Nimrod/O and the parent process of DEMO and, in use it translates data formats and requests between these two applications (Figure 6-2). The user may alter specifics of the DEMO optimisation by editing DEMO's initialisation file. For the convenience of the user, fields in Nimrod/O's plan file that are repeated in DEMO's initialisation file are automatically inserted into DEMO's initialisation file by the interface before it spawns DEMO.

The *stopping criterion* for DEMO is specified in its initialisation file as a maximum number of candidate evaluations. Once this limit is reached, DEMO writes the current Pareto front to a file called `fronts.out`. Further files such as the statistics on the population's evolution and a log file are written by DEMO before it terminates. The DEMOinterface also detects when the maximum number of evaluations has been reached and notifies Nimrod/O which likewise finalises its files and terminates.

In addition to creating the DEMOinterface, contributions of this author are; the above alterations to DEMO, and, further developing Nimrod/O for multi-objective compatibility. The rest of this chapter concerns testing the solution by minimising a two-objective mathematical function, and the three-objective shape optimisation of an engineering part using the Finite Element package, Code_Aster.

## 6.3 Experimental set up - Poloni test function

Poloni's function [93] offers a convenient way to test the DEMO algorithm. It is a two parameter, two response, mathematical function (6-2).

$$x, y \in [-\pi, \pi]$$

$$f_1(x,y) = -\left[1 + (A_1 - B_1)^2 + (A_2 - B_2)^2\right]$$

$$f_2(x,y) = -\left[(x+3)^2 + (y+1)^2\right]$$

Where,

$$\begin{aligned} A_1 &= 0.5\sin 1 - 2.0\cos 1 + \sin 2 - 1.5\cos 2 \\ A_2 &= 1.5\sin 1 - \cos 1 + 2\sin 2 - 0.5\cos 2 \\ B_1 &= 0.5\sin x - 2\cos x + \sin y - 1.5\cos y \\ B_2 &= 1.5\sin x - \cos x + 2\sin y - 0.5\cos y \end{aligned} \qquad \textbf{(6-2)}$$

Price [88] provides a guide to choosing the population size as $10 \times d$ where $d$ is the number of dimensions of the problem, therefore in this test $d$ = 20. The weight of the mutation scaling factor can be any value in the interval [0,2] and was chosen as $\mathrm{F} = 0.5$. The crossover probability must lie in the interval [0,1] and $\mathrm{CR} = 0.3$ was chosen. These $\mathrm{F}$ and $\mathrm{CR}$ values were used for both of the optimisations presented in this chapter. Price and Storn [94] describe the settings for these parameters in more detail. A concurrency setting of 4 directed

Nimrod/O to perform one function evaluation on each of the four cores of the quad-core host machine at any one time.

```
parameter p float range from -3.1415926 to 3.1415926
parameter q float range from -3.1415926 to 3.1415926
results 2

task main
   copy  poloni node:poloni
   node:execute  ./poloni $p $q
   copy  node:exp_result output.$jobname
endtask

method external "DEMO" ./DEMOinterface
  starts 1
  endstarts
endmethod
```

**Figure 6-3 Nimrod/O plan file: poloni.shd**

## 6.4 Result of the Poloni optimisation

Figure 6-4 shows a scatter plot of the Poloni function. 600 function evaluations were performed by Nimrod/O and the final Pareto set of 20 candidates found by DEMO is shown with superimposed square diamond markers. An interpolated line has been added to aid clarity. Visual inspection of this Pareto set indicates that DEMO has been successful in attaining the two aims of; finding a diverse range of solutions, and, finding solutions that are as close as possible to the ideal Pareto front. Arguably, this front is superior to that obtained by Poloni et al. [93] with their MOGA (Multi-Objective Genetic Algorithm) optimiser which involved 50 candidates and 2500 evaluations.

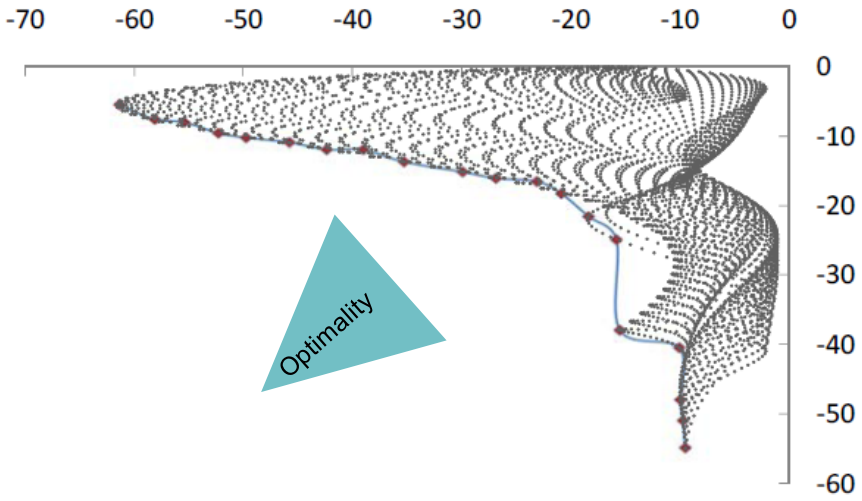



**Figure 6-4 Poloni function, Pareto set superimposed**

## 6.5 Experimental setup – the shape optimisation of a rib-reinforced wall bracket

The shape under consideration is a rib-reinforced wall bracket. The back face of the bracket is constrained and a distributed loading is applied to the protruding face, simulating the bracket supporting a weight of approximately 200kg. Technical drawings (Figure 6-6 and Figure 6-7) show the dimensions of the part (mm) as well as the five decision variables $(A,...,E)$. These variables will be optimised to minimise the three objective functions of: mass, maximum deflection, and, maximum VonMises stress. Minimising the mass conflicts with minimising the stress and the deflection and so the problem will not reduce to one optimal solution – instead a Pareto set of solutions will be found.

| **Table 6-1 Wall bracket decision variables** | |
|---|---|
| A = Thickness of bracket plate (mm) [1,10] | |
| B = Thickness of ribs (mm) [1,10] | |
| C = Placement of ribs (%). When:<br>C = 0, Rib distribution is widest<br>C = 100 Rib's Inner faces are 10mm from mounting holes | |
| $Absolute\_Offest\_OuterFace_{Rib\_1} = 119 - \left( {}^{C}/_{100} \times (29 - B) \right)$ | **(6-3)** |
| $Absolute\_Offest\_OuterFace_{Rib\_2} = 1 + \left( {}^{C}/_{100} \times (29 - B) \right)$ | **(6-4)** |
| D = $x$ displacement of curve control point [30,70] | |
| E = $y$ displacement of curve control point [30,70] | |

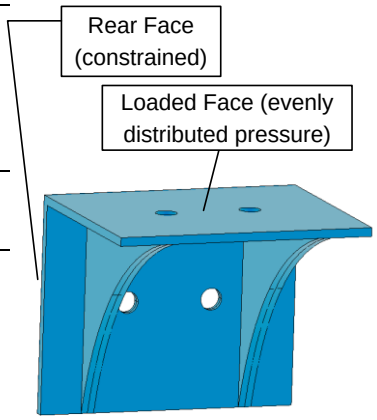



**Figure 6-5 Rib-reinforced wall bracket**

A stand alone computer was used for the results in this chapter with a Quad Core AMD Phenom 2.5GHz processor, 4MB cache with 4GB of RAM installed. The operating system was CAELinux2008 [95] which includes the open-source CAE software: Salomé, Code_Aster, Code_Saturne and OpenFOAM. For this work, only Salomé and the Finite Element software of Code_Aster were used. Onto the base installation of the operating system, the source codes for Nimrod/O 2.9, DEMOinterface and DEMO were compiled and installed.

The five decision variables in Table 6-1 comprise the thickness of the bracket, A, and the thickness of the ribs, B, a distribution of the ribs, C, and co-ordinates for a curve control point D and E. The distribution of the ribs is presented to the optimiser as a floating point variable in the range [0,100], however this variable needs to be translated into physical dimensions on the bracket itself. The equations used to translate the variable C are equations (6-3) and (6-4)**.** These equations are necessary to accommodate changes to the rib thickness, B, and guarantee that when C=100 the inner faces of both ribs will be exactly 10mm from the centre of the mounting holes irrespective of the value B (Figure 6-6). Likewise, when C=0, the outer faces of the ribs will be located at their widest distribution: 1mm from the outer edges of the bracket itself. Both ribs are symmetrically distributed. D and E are $x$ and $y$ co-ordinates of a point through which the profile of the ribs is interpolated. D and E are in the interval [30, 70], the 30 being the displacement in mm from the inner face of the bracket therefore keeping the rib profile point independent of A (Figure 6-7)**.**

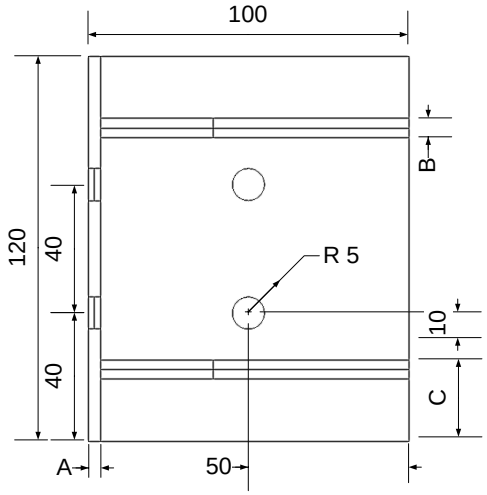



**Figure 6-6 Plan view of the wall bracket**

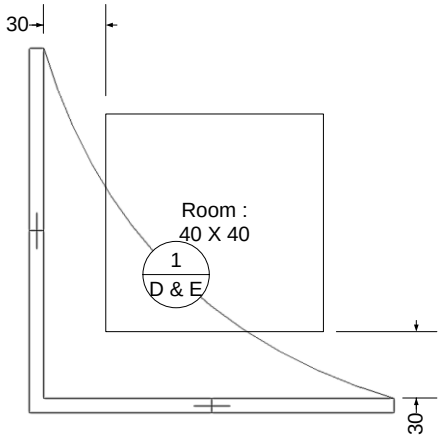



**Figure 6-7 Side elevation of the wall bracket**

### 6.5.1 The shape optimisation job.

The flow chart, Figure 6-8, shows the steps involved for shape optimisation using Code_Aster, Nimrod/O and DEMO. The first two steps involved setting up the shape and the optimisation, but the main work was conducted in an automated loop governed by Python scripts and simple shell scripting.

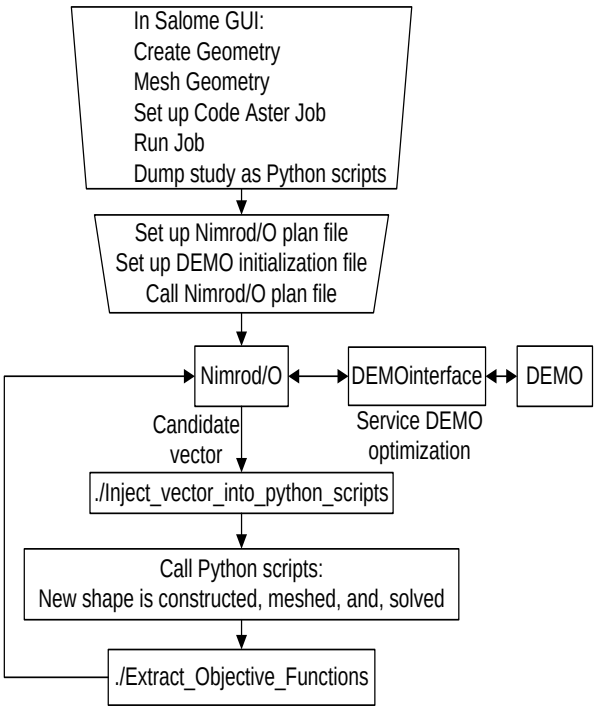



**Figure 6-8 Flowchart of the shape optimisation process**

From the Graphical User Interface (GUI) of Salomé, arbitrary settings for the decision variables $(A, \ldots, E)$ were chosen in building the body of the first shape. The geometry was auto meshed with the in-built algorithms shown in Table 6-3

The volume contained ~ 11,000 tetrahedrons after meshing. The Code_Aster Linear Elastic job was set up with a distributed pressure loading of 0.16667 MPa that represents ~200kg mass on to the upper surface of the bracket. The degrees of freedom for the rear face and interior of the rear bolt holes is given by (DX,DY,DZ) = (0,0,0). The relevant physical properties of the chosen material, Plain Carbon Steel, are given in Table 6-2. After verifying a successful

run of the Code_Aster solver, the above three steps were “dumped as Python study”. In this way templates were created that could later be called from the command line.

**Table 6-2 Material properties of the wall bracket**

| Plain Carbon Steel | |
|---|---|
| Young's modulus, E (Gpa) | 200 |
| Poisson's ratio, $\upsilon$ | 0.3 |
| Density $\left(\mathrm{g}/\mathrm{cm}^3\right)$ | 7.86 |
| Yield Stress, $\sigma_y$ (MPa) | 280 |

**Table 6-3 Auto-meshing settings**

| Meshing | Applied Algorithms | Applied hypotheses |
|---|---|---|
| 1D | Average length (6) | Wire discretisation<br>Added: Quadratic Mesh |
| 2D | MEFISTO_2D | Length from edges |
| 3D | Tetrahedron (Netgen) | |

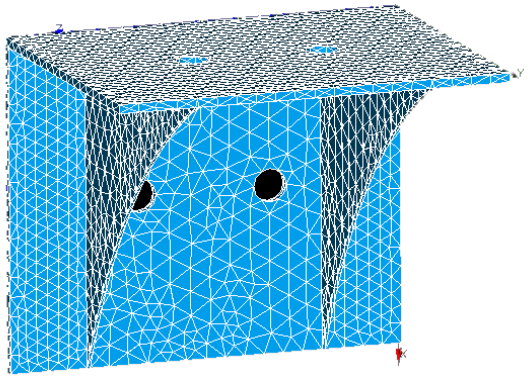


**Figure 6-9 The auto-meshed wall bracket**

Two edits were then necessary in the text files *name*`.comm` and *name*`GEMO.py`. In the *name*`.comm` text file, maximum deflections and principal stresses were requested to be included in the plain text *name*`.resu` results file of Code_Aster. In the Python geometry script, *name*`GEMO.py`, the following lines were added adjacent to the last line:

```
myTuple = geompy.BasicProperties(finished_body)

myMass = (myTuple[2]/1000) x 7.86
```

This calculates the volume of the shape and multiplies by the density. Further Python commands save this mass to file. Two simple C++ programs were also

written. `Inject_vector_into_python_scripts` takes the decision variables $(A, \ldots, E)$ as arguments, parses the template of *name*`GEMO.py`, and inserts changes to the geometry script at runtime. `Extract_Objective_Functions` is called after Code_Aster, extracting the calculated values for maximum deflection and the maximum VonMises stress from *name*`.resu`. The mass is also read-in from file, and the three objective functions are then formatted for Nimrod/O by `Extract_Objective_Functions` and saved to file. After setting up Nimrod/O's plan file and DEMO's initialisation file, a small number of shell scripts were created to implement automation. The memory requirement for an individual job was ~1.3GB. With the installed 4GB of RAM, and with the operating system overhead, a concurrency setting of 2 was the maximum level of parallelism attainable without paging to the hard disk. 6GB or more of RAM would have permitted four concurrent shape evaluations.

## 6.6 Results of the shape optimisation of the rib-reinforced wall bracket

800 candidate evaluations were performed by Nimrod/O, each involving the creation of new geometries and a linear elastic simulation by Code_Aster. The population size was N=50 and four results from the final Pareto set are given in Table 6-4. Across the final, 50 candidate Pareto set, the decision variables fell in the intervals: A[1.0,10.0] B[1.0,5.58] C[86.6,97.7] D[30.0,67.0] E[30.0,57.1]

The full Pareto set is plotted in the 3D scatter graph, Figure 6-10, showing mass, maximum VonMises stress and maximum deflection on each axis. In Table 6-4, displayed are the two heaviest candidates among the Pareto set for which calculations of maximum VonMises stress and maximum displacement were least. The lightest candidate was found to have a maximum VonMises stress of only 3% below the $\sigma_y$ of 280MPa. A typical safety-factor setting of 3.0 would exclude this bracket from use, and likewise the next 16 light-weight solutions due to high imposed stresses. By inspection of the scatter graph in

Figure 6-10, there is a region containing a small number of candidates (lying near the point where the mass begins to increase significantly) that substantially reduce the stress and deflection when compared to the lightest candidates. One of these is labelled the “compromise solution” (Table 6-4). For this candidate, the maximum calculated VonMises stress is 15.2% of the $\sigma_y$ and the Mass is only 28.3% of the two heaviest solutions. The deflections of this compromise solution are represented visually in Figure 6-11**.** The greatest deflections of this solution are located in the 50% of the loaded face that is furthest from the back plate, at the extreme left and right edges.

**Table 6-4 Results of the multi-objective wall bracket optimisation**

| Decision variables | | | | | Objective functions | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Max Deflection (μm) | Max VonMises (MPa) | Mass (kg) | |
| A | B | C | D | E | | | | |
| 1.00 | 1.00 | 91.1 | 30.0 | 33.0 | 739 | 271 | 0.22 | Least Mass |
| 10.0 | 4.94 | 97.7 | 40.1 | 30.0 | 0.90 | 3.82 | 1.99 | Least Stress |
| 10.0 | 5.44 | 91.7 | 33.8 | 30.0 | 0.81 | 3.90 | 1.98 | Least Deflection |
| 2.71 | 1.00 | 90.4 | 40.1 | 41.4 | 29.4 | 42.5 | 0.56 | Compromise solution |

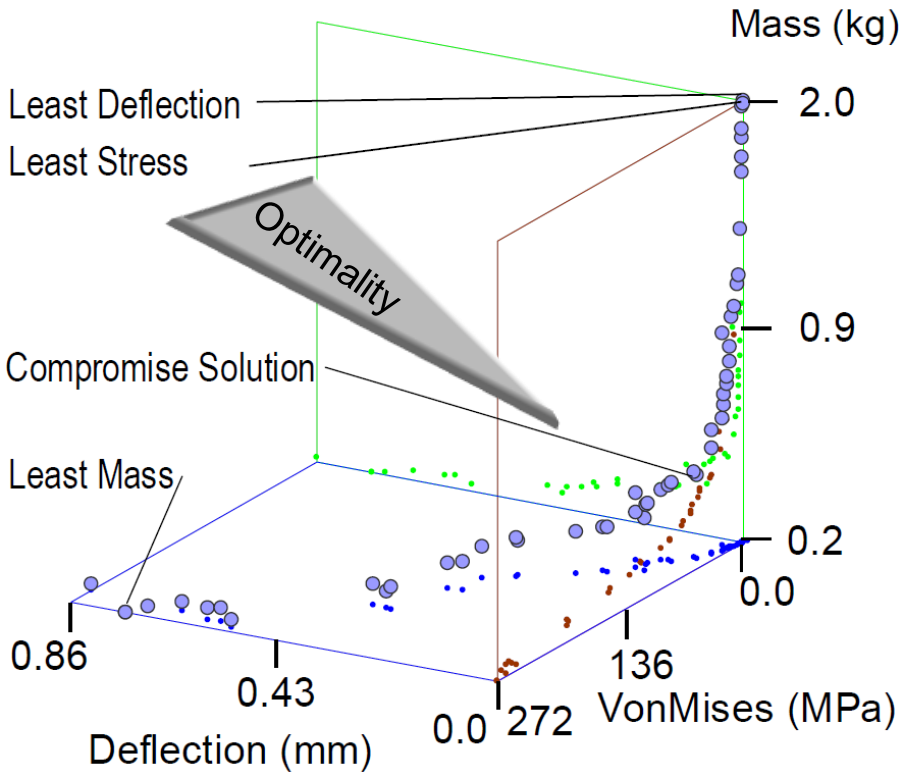



**Figure 6-10 3D scatter plot of the Pareto set in the objective space**

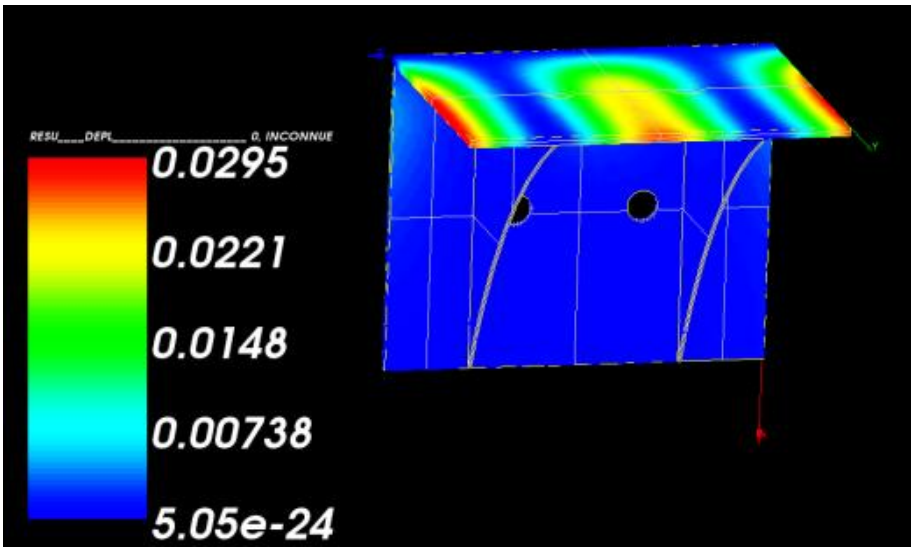



**Figure 6-11 Deflections of the compromise solution (key in mm)**

## 6.7 Conclusion

This chapter has described the successful implementation of the DEMO - Nimrod/O interface and illustrated its usage with two, truly multi-objective, optimisations. A parameter that enables concurrent candidate evaluations has been implemented which can reduce the wall-clock time for optimisations when multiple processors are available, or, be tuned-out by the user - potentially accelerating the convergence to the Pareto front.

Addendum: An early intent was to have a fifth main chapter of work. In this, the Cascade Correlation metamodel would have been applied to accelerate the wall bracket optimisation of this chapter, and the aerofoil optimisations of Chapter 5. In this way, any enhancement offered by the metamodel could be evaluated objectively. For both case studies, the training of the surrogate resulted in such high errors that it was of no real use in the shape optimisations. Arguably, this was an artifact of the high modality of the response surfaces of these two problems.

# 7 CONCLUSION

The central theme of this thesis was to explore, develop, or enhance methods of reducing the computational load of optimisation with a particular focus on multimodal functions. Reducing the computational load of optimisation was first approached by building a neural network based surrogate model. In order to validate the performance of such a surrogate, multimodal mathematical test functions were used (Chapters 3 and 4). As these test functions are those normally applied to test optimisation algorithms, they formed an appropriate choice for a surrogate model that would ultimately be used to assist design optimisation. Each test function produced a response surface that a capable surrogate model should have been able to mimic accurately.

When using neural networks to map the response surface of any given function, we need to minimise the error constituents of bias and variance. The first step is to find the “Goldilocks” topology; namely a neural network that does not have too few or too many neurons, but just the right number. Too few neurons and the neural network will not possess enough complexity to map the features in our training dataset leading to underfitting, i.e. high bias. Too many neurons and we will have too much complexity; we will have endowed the neural network with an ability to fit to noise and we will lose generalisation of the underlying function. High variance (overfitting) will result. Finding the Goldilocks number of neurons becomes a necessity for achieving a good fit when using any type of neural network, and yet there is no pragmatic approach for determining this number a priori.

Ostensibly, the Cascade Correlation neural network forms the ideal basis for universal function approximation. There is no requirement to tune any training parameters unlike, say, backpropagation training where choices of *momentum factor* or *learning rate* impact the quality of the learning. The Cascade Correlation neural network begins training with no hidden neurons (i.e. empty). We will have very high bias and no ability to overfit – low variance. The network

grows in size by adding and training single neurons with each training step. Our bias falls as the neural network attains more ability to fit to the underlying function. Given that we will have ensured that our training will halt automatically (by using an implementation of early stopping) the optimal number of neurons should have been found in all cases. Without an early stopping mechanism, too many neurons will be added and the error will increase due to overfitting. As seen in Chapter 3, the use of early stopping is essential and it also delivers the benefit of a reduction in training time. Variance can be further minimised by ensembling, as variance is inversely proportional to the number of neural networks in an ensemble. Combining the techniques of early stopping and ensembling was found to reduce the error of Cascade Correlation neural networks by a factor of 2.8. There is, however, a training penalty with ensembling as training times are directly proportional to the size of the ensemble.

Chapter 3 has contributed to the knowledge-base of Cascade Correlation users in a variety of ways. Firstly, by determining the benefits offered by different sizes of early stopping sets and, secondly, by offering an alternative to employing a testing dataset; namely the use of a sufficiently large early stopping set as a proxy for a testing dataset. Determining the optimal amount of samples needed to train efficiently with this neural network was a third contribution. This was found to be proportional to the dimensions of the problem; the optimal amount being around 100 samples per dimension, the minimum 32. However, the fact that such a value could even be determined is the first sign of this neural network's weakness. Why should we be able to find such a relationship? It should be the complexity of the problem at hand that governs how many samples are required to describe its features.

Chapter 4 uncovered the limiting feature of Cascade Correlation; *the bias problem* of Cascade Correlation neural networks was postulated to be due to the weight freezing mechanism inherent in the algorithm. The standard version of Cascade Correlation has been used throughout this work, and so no changes

were made to remove weight freezing. This neural network's weight freezing problem was overcome by subdividing highly multimodal datasets into *patches* with one neural network ensemble trained per-patch. The patchworking algorithm represents another contribution of this work. Seen in Chapter 4's table of results and clearly visible in the surface plots, patchworking significantly improved the performance of Cascade Correlation on multimodal functions.

Surrogate (or meta) modelling is a widely published method for accelerating optimisations; especially those for which the evaluations of objective functions are very computationally expensive. Over 12 months of C++ code development, and the 15,000 lines of code that this represents, did result in a functioning Cascade Correlation-based metamodel. This is a metamodel that can be, and in testing has been, integrated with the optimisation toolkit of Nimrod/O. The latency of querying this metamodel is wholly independent from the time taken to first generate the objective functions upon which it was trained. In all cases, a query for the evaluation of an objective function is returned in less than 10ms. Hence, after training this metamodel (and validating that training as successful) 120,000 different designs can be evaluated by an optimiser in less than 20 minutes – a significant speedup.

The aerofoil case study of Chapter 5 determined an appropriately small search domain for NACA 4-digit aerofoil optimisation such that other researchers may reduce the computational load of similar, low Reynolds number, optimisations. Although, the contribution of Chapter 5 is only minor, there are directly measurable outcomes; namely the two MSc Theses that further explored the research gaps revealed by this case study.

Chapter 6 consisted of a significant contribution to the optimisation community. Although the two software packages, Nimrod/O and DEMO, were already in existence, they were combined together for the first time. Nimrod/O was already part of a suite of tools that could distribute problems over a Grid, or cluster, of computers and thereby share the computational load of optimisations. The contributions lie in enabling true multi-objective optimisations for the first time,

and enabling parallelism in DEMO. Again, there are outcomes to-date. In collaboration with Dr. Timos Kipourous another multi-objective optimiser (NSGA) has been integrated with the latest version of Nimrod/O and, in a private communication, it is believed that he also plans to integrate MOTS (Multi-objective Tabu Search) in the near future.

## 7.1 Final words

Whilst a cascading topology neural network remains a very workable solution to the “Goldilocks” problem of neural networks, Cascade Correlation is encumbered with a weakness at mapping multimodal functions that was found only in the later stages of this author’s research. The patchworking solution, contributed here, has been shown to overcome this weakness but only by increasing exponentially the demand for training samples. Such is this exponential increase that any speedup attained from metamodelling is ameliorated by the slowdown caused by first having to gather such vast numbers of training samples. Hence, we would prefer not to have to use patchworking. This author’s evaluation of the Cascade Correlation neural network leads to the ultimate conclusion of this thesis:

Only if expert knowledge can give the assurance that the training data is of low modality can we have confidence in applying Cascade Correlation neural networks for surrogate modelling. Hence, in the general case, this neural network type should not be relied upon for surrogate modelling roles.

# 8 FURTHER WORK

This chapter should first begin with the future planned work of the current author. That work pertains to the interfacing of DEMO with Nimrod/O. As stated in Chapter 6, the functionality of the elitism mechanism in DEMO is uncertain. To be assured that it is enabled correctly requires re-visiting the C++ code. When this work is undertaken, an additional feature will be implemented; namely a re-start mechanism. Whilst both Nimrod/O and DEMO have a caching mechanism, DEMO has to be started afresh after an optimisation has halted unexpectedly. Given that DEMO can already write to a log-file the candidates of the Pareto front for every generation, it should be a relatively simple coding task to enable the parsing of old log files - thereby instantiating a new optimisation from the last known good-population prior to a crash.

With respect to future work that may interest other readers, the topics of research pertain to the neural network studies of Chapters 3 and 4. An implementation of Constructive Back Propagation (CBP) would form the basis for useful research. According to the literature, this neural network trains just as rapidly as Cascade Correlation but, due to CBP's inherent ability to train multiple neurons at each time step, CBP may well learn to approximate highly multimodal surfaces that presently lead to failures for Cascade Correlation neural networks. The CBP literature only considers training two neurons at a time; hence there is a research gap to examine the effects of training more than two neurons. It could be speculated that an adaptive mechanism could be implemented; one that scales up and down the number of neurons added at each training step. We recall that too many, or two few, neurons form the basis for a neural network with poor predictive qualities. Such an adaptive mechanism would scale up and down the count of neurons that it trains for each layer by tracking the progression of the neural network's error against a validation dataset.

# REFERENCES


1. *Machine Learning Repository*. [Available from: http://archive.ics.uci.edu/ml/].
2. Khandani, S., *Engineering design process - Education transfer plan*. 2005: Valley College, Pleasant Hill California.
3. Abramson, D. et al. *An Automatic Design Optimization Tool and its Application to Computational Fluid Dynamics*. in *Proceedings of the Super Computing 2001 Conference, Denver, USA*.
4. Wang, G.G. and Shan, S., *Review of Metamodeling Techniques in Support of Engineering Design Optimization.* ASME Transactions, Journal of Mechanical design, 2007, 129(**4**): p. 370-380.
5. *Release 11.0 Documentation for ANSYS Chapter 3.5: Probabilistic Design Techniques*. [Available from: http://www.kxcad.net/ansys/ANSYS/ansyshelp/Hlp_G_ADVPDS5.html].
6. Keane, A.J. and Prasanth B., *Computational Approaches for Aerospace Design*. 2005: John Wiley & Sons, Ltd.
7. Xu, H., *An Algorithm for Constructing Orthogonal and Nearly-Orthogonal Arrays With Mixed Levels and Small Runs.* Technometrics, 2002. (**44)**: p. 356-368.
8. Sloane, N.J.A., *A Library of Orthogonal Arrays* [Available from *http://www2.research.att.com/~njas/oadir*].
9. Chantrasmi, T., A. Doostan, and G. Iaccarino, *Padé–Legendre approximants for uncertainty analysis with discontinuous response surfaces.* Journal of Computational Physics, 2009. **(228)**: p. 7159-7180.
10. Dubois, D. and Prade, H., *Fuzzy sets and systems: Theory and applications*. 1980: Academic press inc.
11. Simpson, T.W. et al., *Metamodels for computer-based engineering design: Survey and recommendations.* Computers, 2001. **(17)**: p. 129-150.
12. Zentner, J.M., *A Design Space Exploration Process for Large Scale, Multi-objective Computer Simulations*. 2006, Georgia Institute of Technology.
13. Emmerich, M.T.M. et al., *Single- and Multi-objective Evolutionary Optimization Assisted by Gaussian Random Field Metamodels.* IEEE Transactions on Evolutionary Computation, 2006. **10**(4): p. 421-439.
14. Koch P.N. et al., *Statistical Approximations for Multidisciplinary Design Optimization: The Problem of Size.* Journal of aircraft, 1999. **36**(1).
15. Balazs, G., *Cascade-Correlation Neural Networks: A Survey*. 2009, Department of Computing Science, University of Alberta.
16. Kwok, T.Y. and Yeung, D.T., *Constructive Feedforward Neural Networks for Regression Problems: A Survey*. 1995, Hong Kong University of Science and Technology: Hong Kong.
17. Gorissen, D., *Heterogeneous Evolution of Surrogate Models*. MSc Thesis 2007, Katholieke Universiteit Leuven.

18. Clarke, S.M., Griebsch, H.H. and Simpson, T.W., *Analysis of Support Vector Regression for Approximation of Complex Engineering Analyses.* Journal of Mechanical Design, 2005. **127**(6): p. 1077-1088.
19. Grudic, G., *Nonparametric Learning from Examples in Very High Dimensional Spaces*. PhD Thesis 1997, The University of British Columbia.
20. Bishop, C.M., *Pattern Recognition and Machine Learning*. 2006: Springer.
21. Thayananthan A. et al., *Multivariate Relevance Vector Machines for Tracking*. In *9th European Conference on Computer Vision*. 2005. Austria: Springer-Verlag.
22. Nissen, S., *Large Scale Reinforcement Learning using Q-SARSA and Cascading Neural Networks*. MSc Thesis 2007, Department of computer science, University of Copenhagen, Denmark.
23. Kasabov, N.K., *Foundations of neural networks, fuzzy systems, and knowledge engineering*. 1996: The MIT Press.
24. Bishop, C.M., *Neural Networks for Pattern Recognition*. 1995: Clarendon Press.
25. Fahlman, S.E. and Lebiere C., *The Cascade-Correlation Learning Architecture.* National Science Foundation under Contract Number EET-8716324 and Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976 under Contract F33615-87-C-1499., 1991.
26. Drago, G.P. and Ridella, S., *Convergence properties of cascade correlation in function approximation.* Neural Computing & Applications, 1994. **2**(3): p. 142-147.
27. Hoehfeld, M. and Fahlman, S.E., *Learning with Limited Numerical Precision Using the Cascade-Correlation Algorithm*. 1991, Carnegie Mellon University.
28. Banks, L. et al., *Comparing Methods for Multivariate Nonparametric Regression*. 1999, Carnegie Mellon University: Pittsburgh.
29. Tetko, I.V. and Villa, A.E.P., *An enhancement of generalization ability in cascade correlation algorithm by avoidance of overfitting/overtraining problem.* Neural Processing Letters, 1997(6): p. 43-50.
30. Schmitz, A., *Constructive Neural Networks for Functions Approximation and their Application to CFD Shape Optimisation*. PhD Thesis 2007, Faculty of Claremont Graduate University and California State University.
31. Schmitz, A., Besnard, E. and Hefazi, H., *Automated Hydrodynamic Shape Optimization Using Neural Networks.* Paper presented to Society of Naval Architects and Marine Engineers (SNAME) Annual Meeting, 2004.
32. Schmitz, A. and Hefazi, H., *Constructive Neural Network Ensemble for Regression Tasks in High Dimensional Spaces.* Sixth International Conference on Machine Learning and Applications, 2007: p. 266-273.
33. Maimon, et al., *Data mining and knowledge discovery handbook*. 2005: Springer.
34. Pandya, A.S. and Macy, R.B., *Pattern Recognition with Neural Networks in C++*. 1995: CRC Press.

35. Matignon, R., *Neural Network Modeling using SAS Enterprise Miner*. 2005: AuthorHouse. P. 146-152.
36. Amari, S. et al., *Asymptotic Statistical Theory of Overtraining and Cross-Validation.* IEEE Transactions on neural networks, 1997. **8**(5): p. 985-996.
37. Wichard, J.D., *Model Selection in an Ensemble Framework*, in *WCCI 2006: The IEEE World Congress on Computational Intelligence*. 2006: Singapore.
38. Granitto, P.M. et al., *Neural network ensembles: evaluation of aggregation algorithms.* Artificial Intelligence, 2004. **163**: p. 139 - 162.
39. Maalawi, K.Y. and Badr, M.A., *Design Optimization of Mechanical Elements and Structures: a Review with Application.* Journal of Applied Sciences Research, 2009. **5(2)**: p. 221-231.
40. Coello, C.A., *Twenty years of evolutionary multi-objective optimization: A historical view of the field.* IEEE Computational Intelligence Magazine, 2006. **1**: p. 28-36.
41. Zitzler, E. and Thiele, L., *Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach.* IEEE Transactions on Evolutionary Computation, 1999. **3 (4)**: p. 292-301.
42. Srinivas, N. and Deb, K., *Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms.* Evolutionary Computation, 1994. **2(3)**: p. 221-248.
43. Jaeggi, D.M., et al., *The development of a multi-objective tabu search algorithm for continuous optimisation problems.* in EJOR feature issue on Adaptation of Discrete Metaheuristics for Continuous Optimization, 2008. **185**: p. 1192-1212.
44. Robic, T. and B. Filipic, *DEMO: Differential Evolution for Multiobjective Optimization.* Third International Conference on Evolutionary Multi-Criterion Optimization, 2005. **3410**: p. 520-533.
45. Molga, M. and Smutnicki, C. *Test functions for optimization needs* [Available from: http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf]. 2005.
46. FANN, *Fast Artificial Neural Network Library*. [Available from http://leenissen.dk/fann].
47. Prechelt, L., *Investigation of the CasCor Family of Learning Algorithms.* Neural Networks, 1996. **10**: p. 885-896.
48. Mehrotra, K. and Ranka, S., *Elements of artificial neural networks*. 1996: The MIT Press. p. 130-132.
49. Kuhfeld, W.F., *Orthogonal Arrays Provided as a Service of SAS* [Available from : http://support.sas.com/techsup/technote/ts723.html].
50. Forrester, A., Sobester, A. and Keane A., *Engineering Design Via Surrogate Modelling: A Practical Guide*. 2008: Wiley Blackwell.
51. Beachkofski, B. and Grandhi, R., *Improved Distributed Hypercube Sampling*. in *43rd Structures, Structural Dynamics, and Materials Conference*.
52. Geman, S., Bienenstock, E. and Doursat, R., *Neural Networks and the Bias/Variance Dilemma.* Neural Computation, 1992. **4**: p. 1-58.

53. *Machine learning database (concrete compressive strength)*. [Available from: http://archive.ics.uci.edu/ml/machine-learning-databases/concrete/compressive/]
54. Yeh, I.-C., *Modeling of strength of high performance concrete using artificial neural networks.* Cement and Concrete Research, 1998. **28**(12): p. 1797-1808.
55. Yeh, I.-C., *Analysis of strength of concrete using design of experiments and neural networks.* Journal of Materials in Civil Engineering, 2006. **18**(4): p. 597-604.
56. Nash, W.J et al., *The Population Biology of Abalone (Haliotis species) in Tasmania. I. Blacklip Abalone (H. rubra) from the North Coast and Islands of Bass Strait*. 1994, Sea Fisheries Division.
57. Meyer, D., Leisch, F. and Hornik, K., *Benchmarking Support Vector Machines*. 2002, Vienna University of Economics and Business Administration.
58. *The R Project for Statistical Computing*. [Available from: http://www.r-project.org/].
59. Squires, C.S., Shavlik, J.W., *Experimental Analysis of Aspects of the Cascade-Correlation Learning Architecture*. Machine Learning Research Group Working Paper 91-1, 1991.
60. Kwok, T.-Y. and D.-Y. Yeung, *Objective functions for training new hidden units in constructive neural networks.* IEEE Transactions on neural networks, 1997. **8**(5): p. 1131-1148.
61. Drago, G.P. and S. Ridella, *On the convergence of a growing topology neural algorithm.* Neurocomputing, 1996. **12**(2-3): p. 171-185.
62. Baluja, S. and Fahlman, S.E., *Reducing Network Depth in the Cascade-Correlation Learning Architecture*. Technical Report, School of Computer Science, Carnegie Mellon University. 1994.
63. Pace, R.K.and Barry, R., *Sparse Spatial Autoregressions.* Statistics and Probability Letters, 1997. **33**: p. 291-297.
64. Riley, M.J.W., K.W. Jenkins, and C.P. Thompson. *Improving the Performance of Cascade Correlation Neural Networks on Multimodal Functions* in *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2010*. 2010. London, U.K.
65. Lehtokangas, M., *Modifed constructive backpropagation for regression.* Neurocomputing, 2000. **35**: p. 113-122.
66. Lehtokangas, M., *Modelling with constructive backpropagation.* Neural Networks, 1999. **12**: p. 707-716.
67. Singer, P.W., *Wired for War*. 2010: Penguin Books.
68. *Unmanned Aircraft Systems: The Global Perspective 2009/2010*: Blyenburgh & Co.
69. Carmichael, B.H., *Low Reynolds Number Airfoil Survey*. 1982, NASA Contractor Report.
70. Simons, M., *Model Aircraft Aerodynamics*. 3rd ed. 1994: Argus Books.
71. Arora, J.S., *Introduction to Optimum Design, Second Edition*. 2004: Elsevier Academic Press.

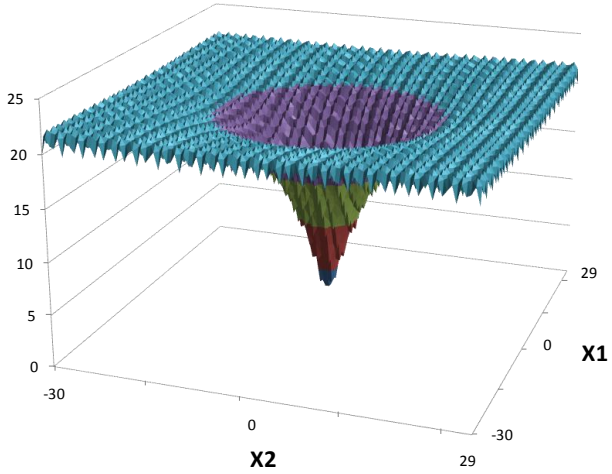
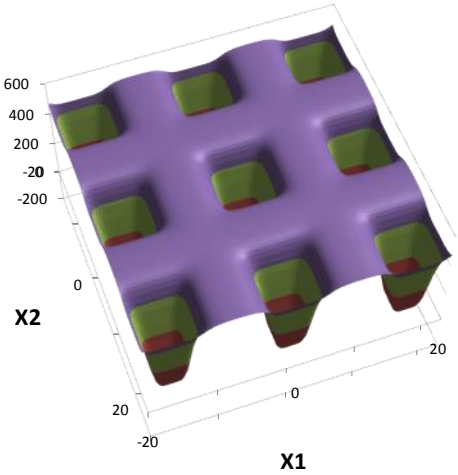
72. Drury, R., *Trajectory Generation for Autonomous Unmanned Aircraft Using Inverse Dynamics*. PhD Thesis, Cranfield University. 2010.
73. Tusar, T., *DEMO Documentation version 1.2*. 2008.
74. OpenCascade, *OpenCASCADE Technology 3D modelling and numerical simulation.* [Available from: http://www.opencascade.org]
75. Spalart, P.R. and Allmaras, S.R., *A One-Equation Turbulence Model for Aerodynamic Flows*. 1992, AIAA Paper 92-0439.
76. Bardina, J.E., Huang, P.G. and Coakley T.J., *Turbulence Modeling Validation, Testing, and Development*. 1997, NASA Technical Memorandum 110446.
77. Launder, B.E. and Sharma, B.I. *Application of the Energy Dissipation Model of Turbulence to the Calculation of Flow Near a Spinning Disc.* Letters in Heat and Mass Transfer, 1974. **1**(2): p. 131-138.
78. Yakhot, V et al., *Development of turbulence models for shear flows by a double expansion technique.* Physics of Fluids A, 1992. **4**(7): p. 1510-1520.
79. Menter, F.R., *Zonal Two Equation k-ω Turbulence Models for Aerodynamic Flows*. 1993, AIAA Paper 93-2906.
80. Baughn, J.W.et al., *Local Heat Transfer Downstream of an Abrupt Expansion in a circular Channel with Constant Wall heat Flux.* Journal of Heat Transfer, 1984. **106**: p. 789-796.
81. Lednicer, D. *The incomplete guide to airfoil usage*. [Available from: http://www.public.iastate.edu/~akmitra/aero361/design_web/airfoil_usage.htm].
82. Chapman, N., *Grid-Based CFD Optimization*. MSc Thesis, Cranfield University. 2009.
83. Debreuil, P.-E., *Automated Design Optimisation*. MSc Thesis, Cranfield University. 2009.
84. Enticott, C. et al., *Electrochemical Parameter Optimization Using Scientific Workflows*, in *IEEE Sixth International Conference on eScience*. 2010: Brisbane, Queensland Australia.
85. Peachey, T.C., *Nimrod/O User Manual*.
86. EnFuzion by axceleon, [Available from : *http://www.axceleon.com/*]
87. Abramson, D., Giddy, J. and Kotler, L., *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?* International Parallel and Distributed Processing Symposium (IPDPS), 2000: p. 520-528.
88. Price, K.V., *Differential evolution vs. functions of the 2nd ICEO.* IEEE Conference on Evolutionary Computation, 1997: p. 153-157.
89. Holland, J.H., *Adaptation in Natural and Artificial Systems*. 1975: The University of Michigan Press.
90. Zitzler, E., Deb, K. and Thiele, L., *Comparison of multiobjective evolutionary algorithms: Empirical results.* Evolutionary Computation, 2000. **8**: p. 173-195.
91. Robic, T., *Performance of DEMO on new test problems: A comparison study.* In Proceedings of the Fourteenth International Eletrotechnical and Computer Science Conference, 2005: p. 121-124.

92. Huband, S., et al., *A scalable multi-objective test problem toolkit.* In Evolutionary Multi-Criterion Optimization (EMO 2005), 2005: p. 280-295.
93. Poloni, C. et al., *Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics.* Computational Methods in Applied Mechanical Engineering, 2000. **186**: p. 403-420.
94. Price K., Storn R. and Lampinen, J., *Differential Evolution: A Practical Approach to Global Optimization*. 2005: Springer-Verlag New York.
95. CAELinux, [Available from :*http://www.caelinux.com/CMS*].

# APPENDICES

## Appendix A Mathematical test functions

### Table A-1 Mathematical test functions

| Function Name | | Range |
|---|---|---|
| | $= -20 \cdot \exp\left(-\frac{1}{5} \cdot \sqrt{\frac{1}{n}\sum_{j=1}^{n} x_j^2}\right) - \exp\left(\frac{1}{n} \cdot \sum_{j=1}^{n} cos(2\pi x_j)\right) + 20 + \exp(1)$ **(A-1)** | $-30 \leq x_j \leq 30$<br>$j = n\ variables$ |
| Ackley |  | |
| | $= \left(0.002 + \sum_{i=1}^{25} (i + (x_1 - a_{1i})^6 + (x_2 - a_{2i})^6)^{-1}\right)^{-1}$<br>where<br>$\begin{pmatrix} a_{1i} \\ a_{2i} \end{pmatrix} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \dots & 32 & 32 & 32 \end{pmatrix}$ **(A-2)** | $-20 \leq x_j \leq 20$<br>$j = 1,2$ |
| De Jong's 5th |  | |

$$= \sum_{i=1}^{5} c_i \exp\left(-\frac{1}{\pi}\sum_{j=1}^{2}(x_j - a_{ij})^2\right)\cos\left(\pi\sum_{j=1}^{2}(x_j - a_{ij})^2\right) \qquad 0 \le x_j \le 2,\ j = 1,2$$

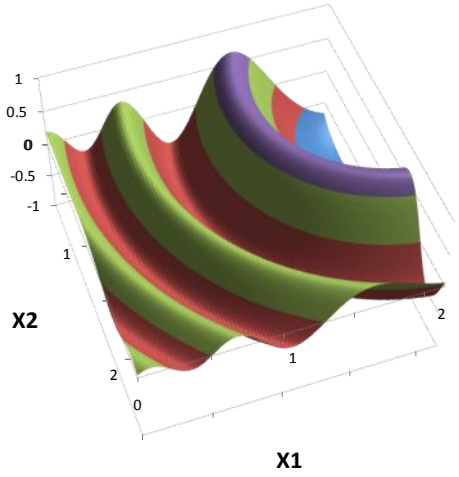
where

$$(a_{ij}) = \begin{pmatrix} 3 & 5 & 2 & 1 & 7 \\ 5 & 2 & 1 & 4 & 9 \end{pmatrix}^{\mathbf{T}} \quad (c_i) = (1\ \ 2\ \ 5\ \ 2\ \ 3)^{\mathbf{T}}$$

**(A-3)**

---

Langermann




---

$$= -\sum_{j=1}^{n} \sin(x_j) \cdot \left(\sin\left(\frac{j \cdot x_j^2}{\pi}\right)\right)^{20}$$

$j = n$ *variables*

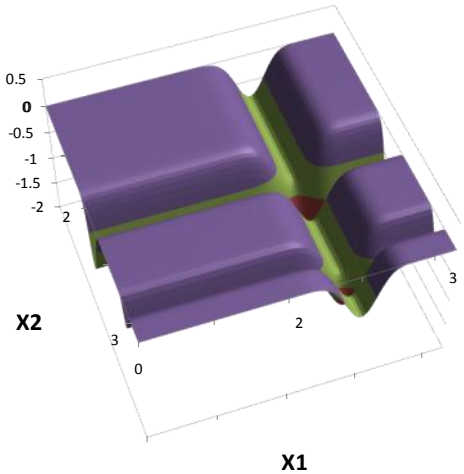
When $j = 2$, $0 \le x_j \le \pi$

When $j = 5$, $1.0 \le x_j \le 1.5$

**(A-4)**

---

Michalewicz



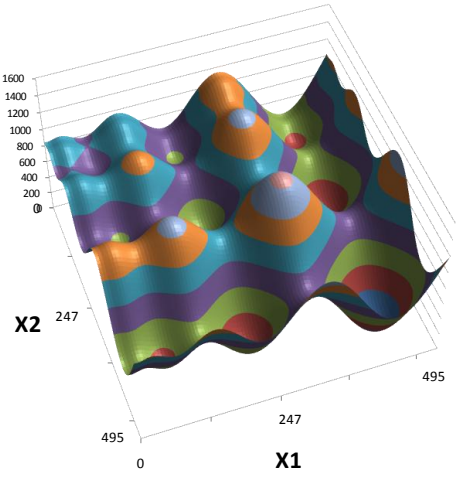
$$= 418.9829n - \sum_{j=1}^{n}\left(x_j \sin\sqrt{|x_j|}\right)$$

$j = n$ *variables*

When $j = 2,\ 0 \leq x_j \leq 500$
When $j = 4,\ 100 \leq x_j \leq 300$
When $j = 5,\ 100 \leq x_j \leq 300$

**(A-5)**

Schwefel

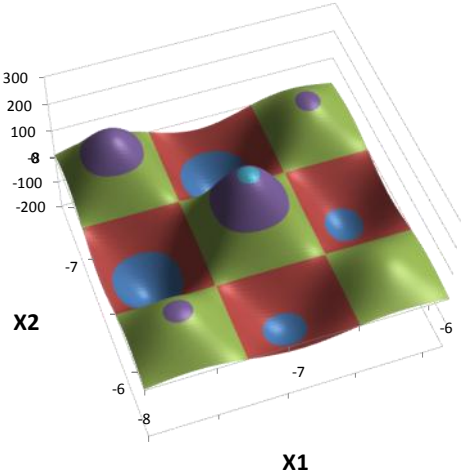



$$= \left(\sum_{i=1}^{5} i\cos\big((i+1)x_1 + i\big)\right) \cdot \left(\sum_{i=1}^{5} i\cos\big((i+1)x_2 + i\big)\right)$$

$-8 \leq x_j \leq -6.2$
$j = 1,2$

**(A-6)**

Shubert

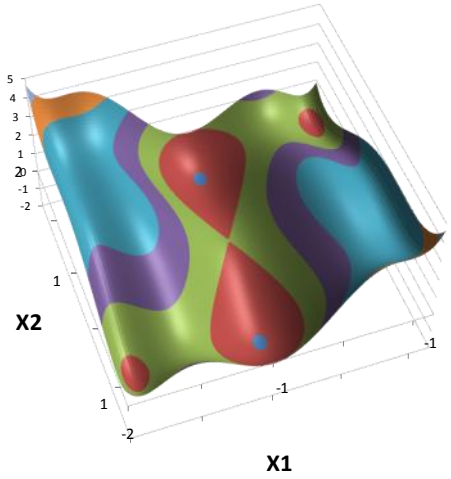


$$= \left(4 - 2.1x_1^2 + {}^{x_1^4}/_3\right) \cdot x_1^2 + x_1 x_2 + \left(-4 + 4x_2^2\right) \cdot x_2^2$$

$-1.9 \leq x_1 \leq 1.9$
$-1 \leq x_2 \leq 1$

**(A-7)**

Six Hump Camel Back




Hartmann, $H_{3,4}$

$$= -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{3} a_{ij}\left(x_j - p_{ij}\right)^2\right]$$

where

| $i$ | $a_{ij}$ | | | $c_i$ | $p_{ij}$ | | |
|---|---|---|---|---|---|---|---|
| 1 | 3.0 | 10 | 30 | 1.0 | 0.6890 | 0.1170 | 0.2673 |
| 2 | 0.1 | 10 | 35 | 1.2 | 0.4699 | 0.4387 | 0.7470 |
| 3 | 3.0 | 10 | 30 | 3.0 | 0.1091 | 0.8732 | 0.5547 |
| 4 | 0.1 | 10 | 35 | 3.2 | 0.0381 | 0.5743 | 0.8828 |

$0 \leq x_j \leq 1$
$\boldsymbol{j = 3\ variables}$

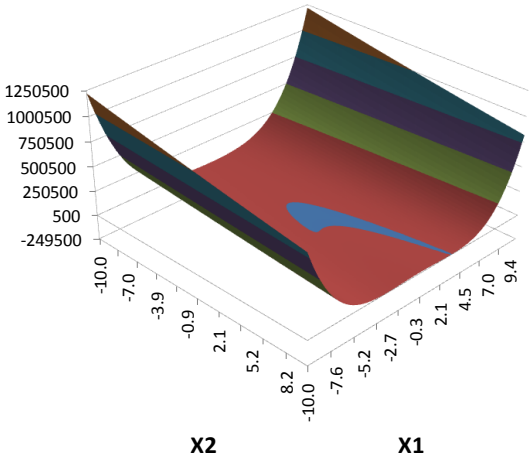
**(A-8)**

$$= \sum_{j=1}^{n-1} \left[100\left(x_j^2 - x_{j+1}\right)^2 + \left(x_j - 1\right)^2\right]$$

$-10 \leq x_j \leq 10$
$j = n\ variables$

**(A-9)**

Rosenbrock



# Appendix B General approach to building a CasCor metamodel

The following steps describe, in order, the approach for building a CasCor metamodel:

- Gather, or generate, sufficient numeric data to train the neural network; the training dataset should number at least 32 samples per dimension, the validation dataset should number at least 5% of this training dataset (or 10 samples whichever the greater) to achieve satisfactory early stopping. Early stopping both increases the quality of the mapping, and reduces the training time and so it should always be used.

- If the user wishes to dispense with a testing set entirely, then consider using a validation dataset of size 30% or more of the training dataset. The MSE calculated on this validation set, can approximate closely the MSE that would be found from using a much larger testing set but without the associated cost of having to generate a large testing set.

- Train one CasCor neural network on this dataset. Inspect Table 3-2 and at this point stop if the MSE is satisfactorily low. Use this CasCor network for metamodelling.

- If the error is unacceptably high; apply ensembling. Arbitrarily choose the size of the ensemble as [7,25] networks. Inspect Table 3-2 again and stop if the MSE of this ensemble is satisfactorily low. Use this CasCor ensemble for metamodelling.

- If the error is still unacceptably high; use the two equations given, (3-3) and (3-4), to determine approximately the contribution of variance and the contribution of bias to this error. If variance is found to dominate: create a larger ensemble. Train larger ensembles until the testing MSE has either reached a suitably low value, or until no further improvement is possible.

If at this point the error is still unacceptably high, bias will be the dominant problem to address. Ensembling and early stopping do not act to reduce bias, however patchworking does. Note that this technique requires training datasets that are exponentially larger than the dimensions of the training dataset.

## B.1 Determining bias and variance by ensembling

An unexpected discovery of the early stopping and ensemble experiments was that the behaviour of a neural network's error can be expressed as a function of the ensemble size. Motivated to determine more precisely the constituents of that error, two equations have been found with which to determine the bias and variance of an ensembled neural network. To the author's knowledge, this particular formulation of bias and variance has not previously been published and could constitute a further contribution of this work.

The accuracy of the calculations of the bias and the variance, when applied as described by the current work, is taken to be of less importance than their approximate ratio as, when $bias^2 \gg variance$, the use of the patchworking technique of Chapter 4 is advocated. Other methods do exist in the literature for the determination of bias and variance. Geman's method [52] proceeds as follows:

- Randomly generate ten training datasets from the whole training data set; each of size N/2 where N is the count of training samples. Call these training datasets $D_1,...,D_{10}$. After training, we have ten neural networks:

$$f^*_{D1}(\mathrm{x}),..., f^*_{D10}(\mathrm{x}) \qquad \textbf{(B-1)}$$

- The ensembled response of these ten networks on the *i*-th example vector $\mathrm{x}_i$ will be: $f_{Ens}(\mathrm{x}_i) = \frac{1}{10}\sum_{j=1}^{10} f^*_{Dj}(\mathrm{x}_i)$ **(B-2)**

- The statistical bias is estimated using unseen data of size, $S$

$$\text{Bias}^2 = \frac{1}{S}\sum_{i=1}^{S}(f_{Ens}(\mathrm{x}_i) - f(\mathrm{x}_i))^2 \qquad \textbf{(B-3)}$$

- The statistical variance of the *j*-th neural network is found from:

$$\text{Variance} = \frac{1}{S}\sum_{i-1}^{S}(f_{Dj}^{*}(\mathrm{x}_i) - f(\mathrm{x}_i))^2 \qquad \textbf{(B-4)}$$

where $f(\mathrm{x}_i)$ is assumed to be the true value of the function that has been approximated.

### B.1.1 STRENGTHS of the current method vs. cross fold

- As can be seen, Geman's cross-fold validation requires a more complex partitioning of any given dataset into different training datasets. The method of the current author is simpler; requiring the use of identical training and validation sets throughout. Also, in the current method, the validation dataset remains unseen throughout and so can be reused to again test the finished ensemble. The same is not true for the cross validation method because such an ensemble of neural networks has seen all the elements of the training and the validation samples, hence no unseen data remains. For this reason, cross-validation necessitates a further dataset if the finished ensemble requires testing. Finally, the cross validation's method of calculating the bias is itself biased and gives slightly higher estimates than the equations presented in this chapter (Geman's assumption is discussed below).

### B.1.2 WEAKNESSES of the current method vs. cross fold

- As stated, the current method requires the validation dataset to remain unseen throughout. As such, the neural networks that constitute any ensemble see a reduced number of unique training samples compared to an ensemble derived from a cross-validation training procedure (which unlimitedly sees all elements of the training and validation samples). Intuitively, this suggests that the current method may produce poorer quality

ensembles than those derived from cross-fold ensembling. Experiments have, however, not been conducted to test for this deterioration.

Geman is careful to state that, for his method, the bias and variance found will only be approximations. Geman's concept of bias could be stated as: "the error of an ensemble of infinite size", or, equivalently "the error in the absence of variance". However, his approximation in calculating bias stems from the assumption that; $EnsSize(100) \approx EnsSize(\infty)$ and $EnsSize(10) \approx EnsSize(\infty)$. Equations (3-3) and (3-4) do not make the same assumption; instead the use of the (EnsSize) and (EnsSize-1) terms are correction factors that acknowledge that bias exists alone *only in the limit*. Compared to German's method, the correction provided by the current work will give reduced estimates of bias and increased estimates of variance that are closer to their true values.

## B.2 Statistical treatment of the Bias and Variance equations

If a comparison between the current method and alternative methods of finding bias and variance was envisaged, there are some relevant issues to be highlighted. What follows is not intended to be a thorough statistical treatment but has been included for the sake of completeness, should further work be conducted by other researchers. Data from the abalone and concrete test cases process will be used as illustrative examples (Table B-1). For convenience, the two equations to be considered are inserted again.

$$Bias^2 = \frac{(EnsSize \times MSE_{Ensemble}) - \overline{MSE}_{EnsSize=1}}{(EnsSize - 1)} \quad \textbf{(3-3)}$$

$$Variance = \frac{\left(\overline{MSE}_{EnsSize=1}\right) - Bias^2}{EnsSize} \quad \textbf{(3-4)}$$

By inspection, we can see that the calculation of bias explicitly uses $\overline{MSE}_{EnsSize=1}$ and $MSE_{Ensemble}$. The calculation of variance explicitly uses $\overline{MSE}_{EnsSize=1}$ and, by including the bias term, implicitly uses the $MSE_{Ensemble}$. There are at two principal ways in which errors in the calculations of bias and variance can propagate. The first is an error in the calculations of the MSEs themselves. Regardless of nomenclature (testing datasets vs. validation datasets), let us re-state that we must test using unseen samples. The evaluation of the MSE is itself an approximation to the integrated MSE over the whole of the neural network(s) response surface(s) and the accuracy of this approximation is inversely proportional to $\sqrt{Count_{UNSEEN}}$ (testing is performed with these unseen samples). The inaccuracy in the calculation of any MSE is also proportional to the standard deviation $(\sigma)$ of each of the errors for every data point in that testing dataset (*standard error,* $SE = \sigma / \sqrt{n}$ where n = number of samples in the testing dataset). The sizes of the testing datasets in Table B-1 are 1253 and 309 which

represent 42% of the size of the training datasets. However, the standard error in the calculations of all the MSEs could be reduced by a factor of 2.4 for the Abalone and by a factor of 4.8 for the concrete were a larger testing dataset used of size = $8\times10^3$.

**Table B-1 Error treatment for the Abalone and Concrete metamodels**

| | Abalone | Concrete |
|---|---|---|
| Dimensions of each problem | 8 | 8 |
| Training set size | 2924 | 721 |
| Validation (and test) set size, 42% of the training set | 1253 | 309 |
| Ensemble size $\left(EnsSize\right)$ | 15 | 15 |
| Mean MSE of the ensemble members $\left(\overline{MSE}_{EnsSize=1}\right)$ | 3.61E-03 | 3.92E-03 |
| MSE of the ensemble $\left(MSE_{Ensemble}\right)$ | 3.44E-03 | 2.87E-03 |
| Standard deviation $(\sigma)$ of the MSEs of the ensemble members | 1.21E-04 | 4.34E-04 |
| Standard error $SE=\sigma/\sqrt{n}$ of the sample mean MSE of the ensemble members, where n = ensemble size | 3.114E-05 | 1.120E-04 |
| Upper 95% confidence limit of the mean MSE of the ensemble members (assumes normal distribution) | 3.67E-03 | 4.14E-03 |
| Lower 95% confidence limit of the mean MSE of the ensemble members (assumes normal distribution) | 3.55E-03 | 3.70E-03 |

The second principal way in which errors in the calculation of bias and variance can arise is due to the use of $\overline{MSE}_{EnsSize=1}$. This is the mean testing MSE of the neural networks that constitute the ensemble. For the concrete compressive strength and the abalone examples, an ensemble size of 15 was used. The standard deviations of $\overline{MSE}_{EnsSize=1}$ is also given Table B-1 and it can be seen that the standard error in $\overline{MSE}_{EnsSize=1}$ is 3.6 times greater for the concrete data than it is for the abalone. Hence, to reduce the standard error in the calculation of $\overline{MSE}_{EnsSize=1}$ for the concrete data to that of the abalone data (at an ensemble size of 15), an ensemble of the concrete neural networks would have to number 195!

### B.2.1 Summary of statistical treatment

If generating training/testing datasets is not expensive, and the primary aim is to evaluate accurately the bias and the variance on particular test functions (for example a comparative study of the performance of different neural network designs) then two new equations have been contributed by the author. A short analysis has shown the necessary care that should be taken when using these equations to best reduce any experimental error. Though building a useful CasCor metamodel would rarely necessitate the use of an ensemble size > 25, ensembles of size > 50 and testing datasets of size > $d \times 10^3$ may be necessary to reduce the standard error. In reducing the standard error, the errors in precisely determining both the variance and bias will also reduce.