Dynamic Modelling Using Genetic Programming

Mark Hinchliffe

A thesis submitted to the Faculty of Engineering at the University of Newcastle upon Tyne in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

Department of Chemical and Process Engineering
University of Newcastle upon Tyne
September 2001

Abstract

Genetic programming (GP) is an evolutionary algorithm that attempts to evolve solutions to a problem by using concepts taken from the naturally occurring evolutionary process. This thesis introduces the concepts of GP model development by applying the technique to steady-state model evolution. A variation of the algorithm known as the multiple basis function GP (MBF-GP) algorithm is described and its performance compared with the standard algorithm. Results show that the MBF-GP algorithm requires significantly less computational effort to evolve models of comparable accuracy to the standard algorithm. The steady-state algorithm is then modified to enable the evolution of dynamic process models. Three case studies are used to demonstrate algorithm performance and show how the MBF-GP algorithm produces performance benefits similar to those observed in the steady-state modelling work. A comparison with neural networks reveals that GP is able to match the accuracy of the network predictions but is more expensive computationally. However, a significant advantage of the GP algorithm is that it can automatically evolve the time history of model terms required to account for process characteristics such as the system time delay.

The model development process is not simply a case of reducing the error between the predicted and actual process output. The parallel nature of GP means that it is ideally suited to solving multi-objective problems. The MBF-GP algorithm is modified to incorporate a Pareto based ranking scheme that allows models to be compared using multiple performance criteria. The ranking scheme allows preference information in the form of goals and priorities to be specified in order to guide the search towards the desired region of the search space. Two case studies are used to demonstrate the performance of this technique. The first example uses the multi-objective algorithm to improve the parsimony of the evolved model structures. The second example demonstrates how a set residual correlation tests can be combined and used as an additional performance measure. In each case, the multi-objective algorithm performs significantly better than the single objective version. In addition, the inclusion of preference information overcomes some of the difficulties associated with conventional Pareto ranking and produces a greater number of acceptable solutions.

Acknowledgements

There are a number of people to whom I am grateful for their contributions made to this thesis. Thanks must be given to friends and colleagues, whether for technical advice, programming ‘expertise’ or moral support. In particular, I would like to mention Mark Willis, Hugo Hiden, Ben McKay, Dominic Searson and Lindsay Parker.

Publications

Some of the work presented in this thesis has appeared or is due to appear in the following publications,

Hinchliffe, M., Hiden, H., Willis, M., McKay, B., Barton, G.W. (1996) 'Chemical Process Systems Modelling Using a Multi-gene Genetic Programming Algorithm'. *Late Breaking Papers, GP '96*, Stanford, USA.

Willis, M., Hiden, H., Hinchliffe, M., McKay, B., Barton, G. W. (1997) 'Systems Modelling Using Genetic Programming'. *Computers and Chemical Engineering*, 21, Suppl., pp. S1161-S1166, Elsevier Science Ltd.

Hinchliffe, M., Tham, M., Willis, M. (1998) 'Chemical Process Systems Modelling Using Multi-Objective Genetic Programming'. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, University of Wisconsin, Madison, Wisconsin, USA.

Hinchliffe, M., Tham, M., Willis, M. (1999) 'Dynamic Chemical Process Modelling Using a Multiple Basis Function Genetic Programming Algorithm'. *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, Florida, USA.

Hinchliffe, M., Willis, M., Tham, M., Montague, G. (2000) 'Dynamic Chemical Process Modelling Using a Multiple Basis Function Genetic Programming Algorithm'. Nineteenth IASTED International Conference, Modelling, Identification and Control, February 14-17, 2000, Innsbruck, Austria

Hinchliffe, M., Willis, M. (2002) 'Dynamic Modelling Using Genetic Programming'. *Proceedings of the 15th IFAC World Congress*, Barcelona, Spain.

Contents

Nomenclature

a	Model parameter
b	Neural network bias parameter
$A(.)$	Output back-shift polynomial
$B(.)$	Input back-shift polynomial
$C(.)$	Noise back-shift polynomial
d	Distance between two population members
D	Kolmogorov-Smirnov test statistic
e	Model residual/noise term
$f(.)$	Function
F_i	Fitness of population member
F	Cumulative probability distribution or Function set
H_0	Null hypothesis
H_1	Alternative hypothesis
$\mathbf{H}$	Hessian matrix
$\mathbf{j}$	Jacobian matrix
k	Current time sample
K	Process gain
$g(.)$	Non-linear function
$\mathbf{g}$	Vector of goal values
G	Number of generations
m	Niche count, number of basis functions
M	Population size
n	Number of process inputs
N	Number of data points
p_c	Probability of crossover
$p(.)$	Parsimony function
p_m	Probability of mutation
p_r	Probability of reproduction
P_i	Performance of population member
q	Back-shift operator
u	Process input variable
s	Laplace operator

s_i	Size of population member
$S(.)$	Empirical cumulative distribution function
$Sh(.)$	Sharing function
t	Generation
T	Terminal set
$\mathbf{u}$	Vector of objective values
$\mathbf{u}^{\overset{\mathbf{u}}{\frown}}$	Components in **u** that violate their goals
$\mathbf{u}^{\overset{\mathbf{u}}{\smile}}$	Components in **u** that satisfy their goals
$\mathbf{U}$	Matrix of process inputs
$\mathbf{v}$	Vector of objective values
$\mathbf{v}^{\overset{\mathbf{u}}{\frown}}$	Components in **v** corresponding to the components in **u** that violate their goals
$\mathbf{v}^{\overset{\mathbf{u}}{\smile}}$	Components in **v** corresponding to the components in **u** that satisfy their goals
w	Cost function or neural network weight parameter
y	Process output variable
$\hat{y}$	Predicted process output
x	Population member, lagged model term or independent variable
α	Learning rate for a neural network
β	Weighting parameter
$\delta(.)$	Kronecker delta function
ε	Model residual
ϕ	Correlation test
Φ	Correlation test objective value
η	Neural network momentum term
φ	Regessors
φ	Number of model terms
θ	Model parameter
Θ	Vector of model parameters
τ	Time shift
τ_p	Process time constant
σ_{share}	Fitness sharing parameter
$\Re$	Ephemeral random constants

1 Introduction

There are many applications across a range of scientific and engineering disciplines that rely on the development of a model of some form. The process industries are no exception, where applications such as inferential estimation, optimisation, process simulation, and process control all depend on accurate models in order to realise their full potential. Two distinct directions can be taken when building a model. The traditional approach is to construct a mechanistic model based on knowledge of the underlying physical and chemical processes. Unfortunately, the system may be extremely complex, making it difficult to derive a model from first principles. In addition, as some aspects of the process may be poorly understood or rely on empirical relationships, the resulting model may not be able to achieve the desired level of accuracy. As a result, the development costs may outweigh any of the benefits to be gained by implementing a model of this type.

The alternative is to develop a data-based model. This technique does not attempt to derive the differential and/or algebraic equations that describe the underlying system. Instead, use is made of available process data to develop a model that relates the target variable (or output) to other process variables (the model inputs). For some applications, a linear model may adequately perform this task. However, the complex and non-linear nature of chemical processes means that more advanced techniques are often needed. A commonly used example of such an approach is the artificial neural network. The use of neural networks has been widespread in recent years due to an abundance of process data and an increase in the availability of relatively low cost computer hardware. The resulting models are more cost effective than mechanistic models as development times are greatly reduced. In addition, data-based approaches are more flexible as the same set of basic tools can generally be applied across a wider range of processes. The conventional approach to empirical model development follows the general path shown in Figure 1-1 (adapted from Söderström and Stoica, 1989).

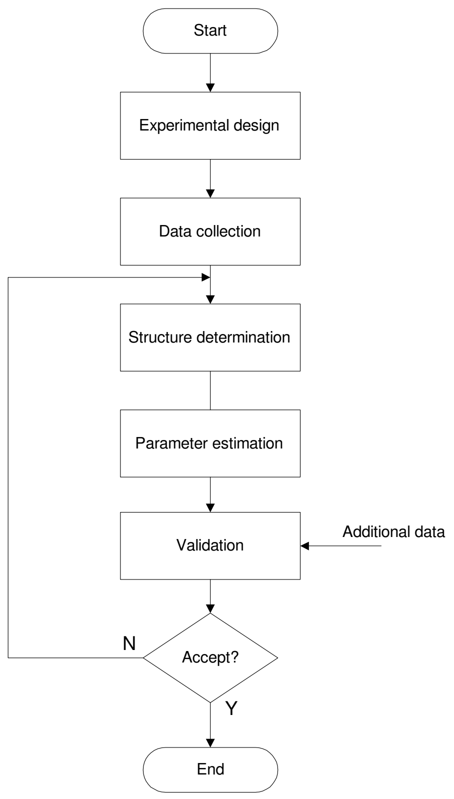


Figure 1-1 – Empirical model building

It can be seen that this is a sequential process with the parameter estimation stage occurring after the model structure has been specified. If the resulting model does not meet the desired performance criteria, the process must be repeated until an adequate model structure is found. Genetic programming (GP) (Koza, 1992) is an evolutionary algorithm that attempts to evolve a set or *population* of solutions to a problem by using the Darwinian concept of natural selection. One of the strengths of GP is its ability to perform optimisation on a structural level. This is an attractive prospect as the algorithm can simultaneously evolve a model's functional form and numerical parameter values. This means that GP has the potential to function as an automatic model building tool. An advantage of this approach is that fewer assumptions have to be made regarding the final form of the model as the algorithm can evolve the model structure from elementary building blocks.

The next stage of the model development process is known as model validation. This process can use a range of different criteria to measure the adequacy of the model. A potential drawback is that the validation criteria are usually applied after the model's structure and parameters have been determined. An area of research that has received a great deal of attention in recent years is that of multi-objective problem solving. The parallel nature of population based algorithms such as GP means that they are ideally suited to this type of problem. One of the potential benefits of using a multi-objective GP algorithm for model development is that additional modelling criteria can be included during the evolutionary process. This approach is different from the traditional method as the three main stages (3,4 and 5 in Figure 1-1) can be dealt with in parallel. Unfortunately, some aspects of model validation (such as the use of unseen data) must be performed after model development and cannot be incorporated into this scheme. However, this approach still has the potential to improve model performance as more design criteria can be considered during the model evolution stage.

The main aims of this thesis are,

- To demonstrate how the modelling performance of the standard GP algorithm can be enhanced by incorporating techniques borrowed from existing system identification theory.

- To describe how GP can be used to automatically develop models of dynamic processes, whilst making few *a priori* assumptions regarding the final model structure.

- To compare the steady state and dynamic modelling ability of GP with a more established data-based modelling technique. Neural networks were chosen for this purpose, as their use has been widespread in the process industries

- To develop a multi-objective GP algorithm that is able to account for additional performance criteria during model evolution.

1.1 Thesis Layout

This thesis is arranged as follows,

- ***Chapter 2*** introduces the main aspects of the GP algorithm, placing particular emphasis on model development. The chapter begins by describing the genetic algorithm (GA) from which GP was originally derived. Chapter 2 also outlines some of the specific features of the algorithm used in this thesis and discusses the rationale behind the main algorithm settings and parameters.

- ***Chapter 3*** demonstrates how GP can be used to develop steady-state models. Aspects of the algorithm that make it different from the standard algorithm described by Koza (1992) are discussed. A different form of the algorithm, known as the multiple basis function (MBF) GP algorithm is introduced and its performance compared with the 'standard' version. This chapter also includes an explanation of the procedures used to compare algorithm performance.

- ***Chapter 4*** Compares the steady-state modelling ability of the MBF-GP algorithm with that of neural networks. This chapter includes an introduction to neural networks and their application to process modelling.

- ***Chapter 5*** describes how the steady-state modelling algorithms can be modified in order to generate and evolve dynamic models. Three case studies are used to compare the MBF-GP and standard GP algorithms. Comparison is also made with filter based neural networks.

- ***Chapter 6*** introduces the relevant aspects of multi-objective evolutionary algorithms, including the benefits of using Pareto methods. The main features of the multi-objective GP (MOGP) algorithm used in this work are described.

- ***Chapter 7*** applies the MOGP algorithm to two dynamic modelling case studies involving additional model performance criteria. The first case study demonstrates how the algorithm can be used to account for the parsimony of the evolved solutions. The second study uses a set of residual correlation tests as a more complex measure of model performance. Comparisons are made between the single objective and MOGP algorithms. The study also highlights the performance gains that can be achieved by using a goal based ranking scheme.

- ***Chapter 8*** summarises the main conclusions resulting from this study and offers some recommendations for future work.

1.2 Thesis Contributions

The main contributions made by this thesis are,

- A detailed assessment of GP algorithm performance on steady-state systems. This includes a comparison between MBF-GP and 'standard' GP algorithms.
- The development of a GP algorithm capable of evolving discrete time models of dynamic systems.
- Dynamic models are represented using a flexible technique that enables the algorithm to identify process characteristics such as the system time delay.
- Comparison is made between GP and artificial neural networks for dynamic and steady-state modelling.
- Algorithm comparisons account for the computational complexity of each approach in addition to the prediction accuracy of the resulting models.
- It is demonstrated how multi-objective GP can be used to account for additional measures of model performance that would normally be considered after the model building process.

- Comparisons are made between the performance of the single and multi-objective algorithms.

- It is shown how preference information in the form of goals and priorities can be used to overcome some of the deficiencies associated with conventional Pareto ranking.

2 Introduction to Genetic Programming

2.1 Introduction

Genetic programming (GP) is an evolutionary algorithm that uses concepts taken from the naturally occurring evolutionary process. The algorithm attempts to evolve solutions by using the Darwinian principle of survival and reproduction of the fittest and genetic operators analogous to those occurring in biological species. GP is a member of a broader class of search and optimisation algorithms inspired by evolution in nature. These algorithms include evolutionary strategies (Rechenberg 1972, Schwefel 1995), evolutionary programming (Fogel *et al.*, 1966, Fogel 1995) and genetic algorithms (Holland 1975, Goldberg 1989). The development of GP was motivated by the desire to enable computers to automatically generate programs. This objective had already been achieved to a limited extent using genetic algorithms. For example, Cramer (1985) described how a genetic algorithm (GA) could be used to evolve simple sequential programs, represented in tree structure form. Fujiki and Dickinson (1987) extended these concepts, describing how LISP source code could be evolved using a GA.

However, Koza (1992) was the first to fully exploit the potential of this approach by developing an algorithm that represents solutions as tree structures using a problem specific syntax. This makes for an extremely flexible technique as the solutions can take a variety of forms. For example, the solutions could be computer programs, mathematical expressions or induction rules. This means that GP has advantages over other algorithms as it can perform optimisation at a structural level. This enabled Koza to demonstrate the application of his GP algorithm to a number of problem domains, including regression, control and classification. Since then, research in this area has grown rapidly and encompassed a wide range of problems. Engineering applications include signal processing (Sharman *et al.*, 1995), electrical circuit design (Koza *et al.,* 1999) and scheduling (Montana and Czerwinski, 1996). Applications with particular relevance to chemical engineering include polymer design (Porter, *et* *al.,* 1996), process controller evolution (Searson *et al*., 1998) and modelling of both steady-state and dynamic processes (McKay *et al*., 1997, Bettenhausen *et al*., 1995). The GP algorithm is an extension of the basic GA, which is described in the next section.

2.2 Genetic algorithms

Holland (1975) described a technique, now known as the GA, that used concepts taken from the naturally occurring evolutionary process to solve problems by performing a highly parallel search. The GA begins by randomly generating an initial set or *population* of candidate solutions. Every population member is allocated a value that is a measure of its performance, known as the individual's *fitness*. A new population is then generated by applying the Darwinian principle of survival and reproduction of the fittest, making use of operators that are analogous to naturally occurring genetic operators such as sexual recombination (crossover) and mutation. The process is repeated over a number of iterations or *generations* in an attempt to evolve increasingly accurate solutions. As the individuals in the GA population are typically stored as fixed length character strings, a suitable encoding scheme must be devised before the algorithm can be applied to a problem. This procedure is outlined in the next section

2.2.1 Problem encoding

An encoding scheme must be developed to provide the algorithm with a way of mapping the points in the problem search space to a character string or other suitable data structure. There must be a way of inverting this transformation, i.e. being able to find the point in solution space that corresponds to a given character string. The most commonly used representation scheme is the fixed length character string inspired by naturally occurring chromosomes. The basic GA uses chromosomes encoded in binary so that individuals consist entirely of strings containing only 1's and 0's. Although this representation is the most common, more advanced schemes can also be used. Examples include the messy GA (Goldberg *et al.,* 1989) and the real-coded or floating point GA (Goldberg, 1990).

The concept of the binary GA string can be illustrated using a function optimisation example. It is assumed that the GA is being applied to a problem involving the optimisation of four numerical parameters,

Parameter values: θ_1 θ_2 θ_3 θ_4

Each parameter is converted into a binary number. In this case, the numerical parameters have been encoded into five digit binary numbers,

Individual genes: | 01001 | 01010 | 10110 | 01111 |

The chromosome is then constructed by concatenating the four subsections or 'genes' to produce a bit-string of 20 characters in length,

Chromosome: 01001010101011001111

The number of bits in each section of the chromosome must be chosen so that the corresponding regression parameter can be represented to the required level of accuracy and vary over the desired range. For example, if a parameter is to be adjusted over the range [-10 10] with a minimum increment of 0.01, the binary string must be able to represent 2000 (20/0.01) distinct numbers. This can be achieved using an 11-bit string, capable of representing 2048 (2^{11}) individual values. The genes do not necessarily have to encode the parameter values in the same way. For example, another gene may use an 11-bit string to represent a number in the range [-1 1] with increments of 0.001. This example highlights the importance of careful problem formulation before attempting to apply the GA. If insufficient thought is not given to the encoding scheme, the GA may have no chance of finding a suitable solution. The application of GAs does not have to be restricted to parameter optimisation problems, with more ingenious encoding schemes allowing the algorithm to tackle a range of problems.

The initial GA population is made from a set of randomly generated individuals by applying a coding strategy similar to that described above. The number of individuals required in the population will depend on the difficulty of the problem and is one of a number of parameters that must be specified at the beginning of the algorithm run. Typical population sizes may be as small as 50 or 100 individuals but harder problems may require larger population sizes to achieve a successful result. The improvement in performance achieved by using larger populations will be at the expense of increased computational effort and a balance between the two must be found.

2.2.2 Fitness Assignment and Selection

The next task performed by the GA is to measure the performance of each population member. This is carried out using a 'fitness' function that assigns numerical values to each of the members in the population. This function will be problem dependent and must be carefully chosen so that it provides an accurate measure of performance for all of the possible solutions that may be encountered during the algorithm run. It is common, although not imperative, to select a function that returns larger values as performance increases so that more 'fit' individuals are assigned greater fitness values. These values are then used to select individuals for breeding in accordance with the principle of survival and reproduction of the fittest. There are a number of possible selection methods, but they all conform to the same general principles,

- Individuals displaying a higher level of performance are more likely to be selected. If it is assumed that fitter individuals are more likely to contain genetic material required to produce a successful solution, it follows that these solutions can be used to generate even fitter solutions.
- The selection method is probabilistic. This means that although the selection procedure is biased towards fitter individuals, there is still a chance that individuals with low fitness values may be selected. This enables the GA to explore parts of the solution space that would otherwise be inaccessible to a hill-climbing approach.

- Reselection is permitted. This is beneficial as it allows fitter population members to be selected more times than less fit individuals, thus promoting the discovery of even fitter solutions.

One example is fitness proportionate selection (FPS) where individuals are selected with a probability that is directly proportional to their fitness, such that,

2-1

Deleted: $P_{S,i} = \frac{\text{Fitr}}{\text{Sum of pc}}$

Where $P_{S,i}$ is the probability of selection for population member i, F is the fitness and M is the population size. This procedure is sometimes referred to as *roulette wheel* selection. Other approaches such as tournament and ranking selection will be discussed in section 2.3.3. The selection technique is used to extract individuals from the population, which are then modified using genetic operators to create the next population. The most commonly used genetic operators are described in the next section.

2.2.3 Application of Genetic Operators

Three basic operators are commonly applied to the selected individuals in order to generate the next population of candidate solutions - direct reproduction, crossover and mutation. The operator is chosen on a probabilistic basis, with each having a different probability of being selected. For example, if the probability of mutation is P_m and the probability of crossover is P_c, the probability of direct reproduction is given by 1-(P_m+P_c). Direct reproduction is the simplest of the genetic operators. The selected individual is left unchanged and is passed straight through to the next population. This provides the algorithm with the ability to preserve fit individuals from one generation to the next.

The crossover operator allows new individuals to be created, with the aim of producing fitter individuals from the genetic material present in existing population members. The process is illustrated using the following parent individuals.

Parent1:	10101011110000
Parent2:	11111100110011

A crossover point (indicated by a vertical line) on each chromosome is randomly chosen using a uniform probability distribution,

Parent 1:	101010|11110000
Parent 2:	111111|00110011

Two new offspring are then formed by transferring genetic material between the two chromosomes.

Offspring 1:	10101000110011
Offspring 2:	11111111110000

These new individuals are then placed in the new population to take part in the next iteration of the algorithm. This recombination operation means that the algorithm is not only able to breed fit individuals together in the hope of producing even fitter solutions, but is also capable of exploring new points in the search space. The form of crossover demonstrated above is referred to as single point crossover. Other forms of crossover are also possible. For example, multi-point crossover transfers several sections of genetic material between individuals.

In the special case where the two parents are identical, the two offspring are also the same, regardless of the chosen crossover point. This occurrence may become a problem if the selection procedure regularly selects the same highly fit members from a population. This may lead to the population becoming dominated by the same individual. If the solution represented by this dominant individual does not meet the necessary solution criteria premature convergence is said to have occurred. This phenomenon is an example of how it can be a disadvantage to rely entirely on crossover. Although the crossover operator usually produces individuals that are different from each other and their parents, the process only makes use of the genetic material present in the current population. One method used to promote diversity and improve the algorithm's ability to exploit different regions of the search space, is to use the mutation operator.

The mutation operator makes a random change to the selected population member. For example, with reference to the following population member,

```
Parent:          1010100010101
```

The first stage of the operation is to randomly select a mutation point (↑) along the chromosome using a uniform probability distribution,

```
Parent:          1010100010101
                        ↑
```

The character at the mutation point is then altered, in this case, a '`0`' is changed to a '`1`'.

```
Offspring:       1010101010101
```

Once mutation has been carried out, the individual enters the new population ready for the next iteration of the algorithm. Typically, the mutation operator is used sparingly, as high mutation rates can have a detrimental effect on algorithm performance. The probability of a mutation operation occurring is sometimes specified on a bit-wise basis. This means that each individual '0' or '1' has a probability, P_m of being mutated. This strategy means that more than one mutation can be applied to any given binary string and can be applied to strings that have been created by crossover. When specified in this way, care must be taken to ensure that P_m is low enough to ensure that mutation is not applied to too many population members. The value will depend on the length of the chromosomes being used, but a typical value would be in the range [0.001 0.01].

The next step taken by the algorithm is to determine which members of the existing population 'survive' and are carried on from one generation to the next. The simplest approach is to generate an entirely new population without preserving any individuals from the previous generation (apart from those selected for direct reproduction).

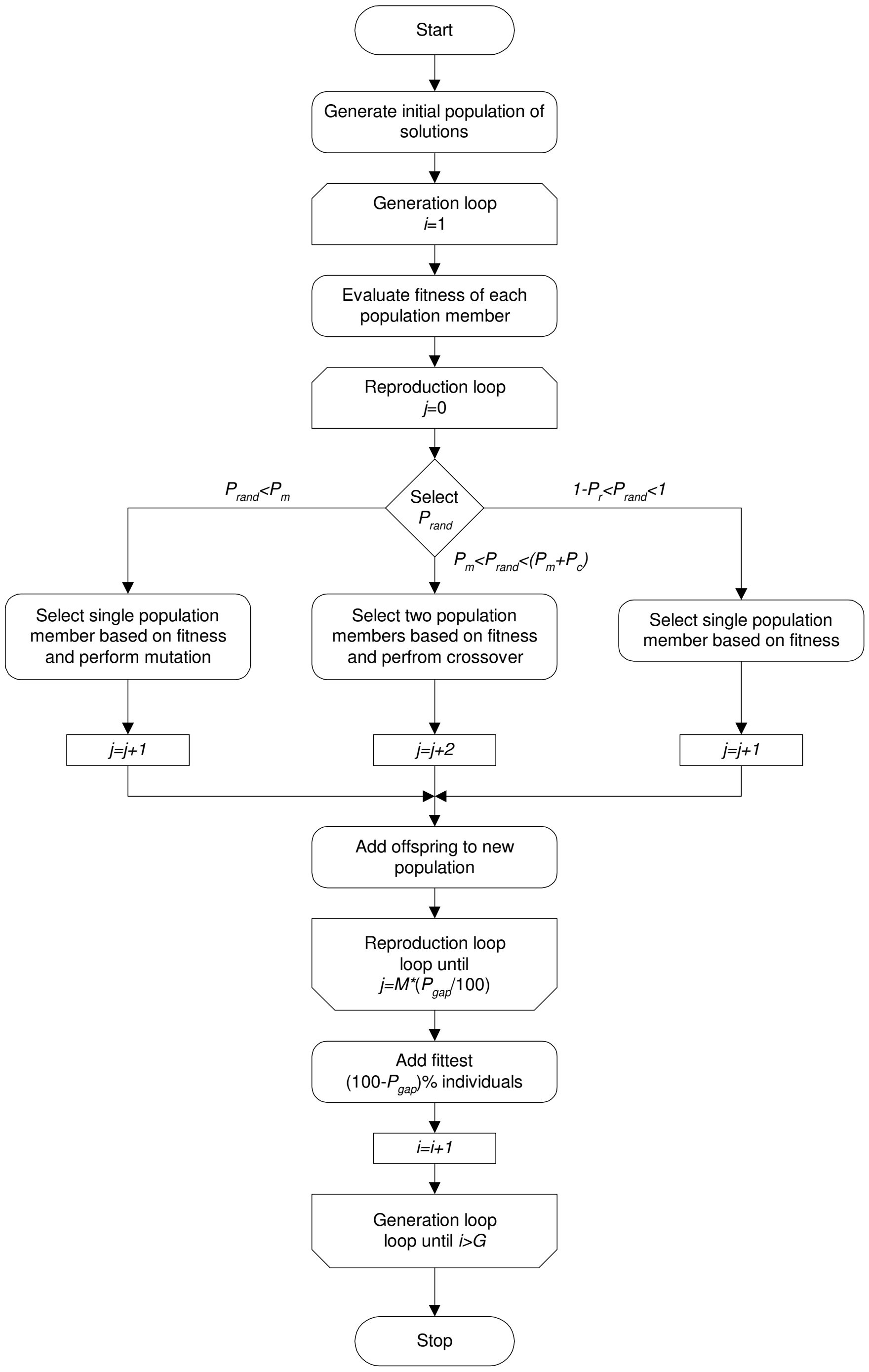

Start
Generate initial population of solutions
Generation loop
i=1
Evaluate fitness of each population member
Reproduction loop
j=0
Select P_{rand}
$P_{rand}<P_m$
$1-P_r<P_{rand}<1$
$P_m<P_{rand}<(P_m+P_c)$
Select single population member based on fitness and perform mutation
Select two population members based on fitness and perfrom crossover
Select single population member based on fitness
$j=j+1$
$j=j+2$
$j=j+1$
Add offspring to new population
Reproduction loop
loop until
$j=M*(P_{gap}/100)$
Add fittest
$(100-P_{gap})$% individuals
$i=i+1$
Generation loop
loop until $i>G$
Stop

Figure 2-1 – Genetic algorithm flowchart

An alternative is to adopt what is known as an *elitist* strategy where the best members of the current population are copied straight over to the next generation. For example, the top 10% of the individuals could be copied over, leaving the remaining 90% of the new population to be created by applying genetic operators. This percentage of new individuals (in this case 90%) is referred to as the *generation gap*.

Once the new population has been created, the processes of fitness evaluation, selection and the application of genetic operators are repeated until some termination criterion is met. This may be a certain fitness level representing a successful solution or a maximum number of generations to be performed. Figure 2-1 is a flowchart of the basic GA algorithm.

2.2.4 Advantages of Using Genetic Algorithms

There are fundamental differences between GAs and traditional optimisation techniques that make them attractive when applied to a wide range of problems. Firstly, the GA can perform a highly parallel search of the solution space. This is only partly explained by the fact that the algorithm uses a population of candidate solutions. Holland's *schema theorem* (Holland, 1975) shows that the GA exhibits a high degree of implicit parallelism by processing sets of unseen individuals similar to those in the current population. This enables the algorithm to discover a wide range of solutions and means that it is less likely to converge around local optima. Additionally, the probabilistic nature of the GA means that the algorithm is not as dependent on the initial starting points. This is in contrast to purely deterministic optimisation algorithms that are much more vulnerable to a poor choice of initial starting conditions. Unlike other optimisation algorithms, the GA does not require any additional problem information such as derivative values as it relies solely on the values provided by the fitness function. This makes the algorithm an ideal choice for problems that prove to be problematic for traditional gradient-based methods due to highly complex or irregular search spaces.

The GA has proven to be a useful tool when applied to a variety of practical engineering problems, including model development. Tan *et al.* (1995) used a GA to develop polynomial ARMAX (Auto Regressive Moving Average with eXogenous inputs) model structures. The technique was rather limited, as the model structure was predetermined, with the GA being used to optimise the model parameters and the maximum time shift of the input. A more general approach was described by Fonseca and Fleming (1996a) who evolved NARMAX (Non-linear ARMAX) model structures using a multi-objective GA. Their algorithm had more influence over the structure of the model, being able to determine features such as the number of model terms, degree of non-linearity and number of lags for the input and output terms. Other process engineering applications include the optimisation of heat exchanger networks (Androulakis and Venkatasubramanian, 1991), process control (Nordvik and Renders, 1991), data analysis (South, 1994) and scheduling (Löhl *et al*., 1998). An excellent introduction to GAs and their application to a variety of problems is given by Goldberg (1989). The next section introduces the fundamental aspects of GP, highlighting the differences between the basic GA and GP algorithms.

2.3 Genetic Programming

Unlike the GA outlined earlier, the GP algorithm uses population members that have chromosomes encoded as tree structures that can vary in size and shape. This approach means that population members can be represented in a form that is specific to the problem being solved. For example, to tackle a regression problem, the population members can be represented as tree structured mathematical expressions. GAs do not have this flexibility as individuals are usually coded as fixed length character strings, meaning elaborate encoding schemes are required to allow their application to a wider range of problems.

Koza (1992) first demonstrated how GP could be applied to regression problems. Traditionally, this task requires the specification of a model structure, followed by the optimisation of the associated numerical parameters in order to achieve the best possible fit. An interesting feature of GP is its ability to perform optimisation on a structural level by evolving populations of tree structured model expressions. This means that GP can be used to optimise the model's functional form and numerical parameters simultaneously. This process is referred to as *symbolic regression*. Two of the most important aspects of the GP algorithm are the function and terminal sets that contain the building blocks used to construct the tree structured population members. These concepts are discussed below, with particular emphasis placed on the application of GP to process modelling.

2.3.1 Terminal and Function Sets

The terminal set contains the elements or variables that are the inputs to the problem. For modelling purposes, the terminal set may simply consist of the process input variables (u_1, u_2,...,u_n). An important part of all regression techniques is the determination of the appropriate numerical parameters that fit the chosen model structure to the output data. One way to enable GP to evolve regression constants is to include a terminal that represents a randomly generated real number. Koza (1992) described these values as *ephemeral random constants* and indicated their presence in the terminal set by the symbol '$\Re$'. If this terminal is selected during tree generation, a new numerical constant is generated and placed in the model equation. The function set is made up of a number of domain-specific functions that, combined with the terminal set, enable the algorithm to construct potential solutions to the problem. For regression, the function set may contain any number of mathematical functions ranging from the basic plus, minus, times and divide operators to functions such as square root, logarithm and exponential. *A priori* knowledge of the problem may be included at this stage by the addition of functions that the engineer thinks may help the algorithm formulate a solution. Figure 2-2 demonstrates how process input terminals can be combined with the mathematical operators in the function set to generate a tree structured model equation ($\hat{y}$ is the predicted process output).

Model equation: $\hat{y} = \log(u_3) + u_1\sqrt{u_2}$

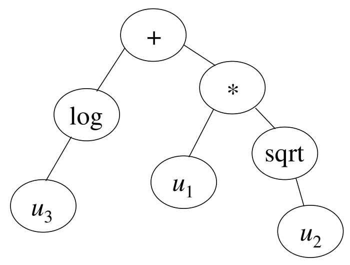


Figure 2-2 – Tree representation of a mathematical expression

The functions must be able to accept values that may be returned by any possible combination of the functions and terminals. This means that the algorithm may have to be supplied with protected versions of some functions. For example, to prevent square roots of negative numbers, the absolute value of the input can be taken, i.e. `SQRT`(x)=$\sqrt{|x|}$. Different applications may require more specialised function sets. For example, the function set may contain functions that perform Boolean operations (e.g., `AND`, `OR, NOT`) and conditional statements (e.g., `if-then-else`).

2.3.2 Fitness function

As with the simple GA described earlier, fitness is a numerical value assigned to each population member in order to measure the performance of each individual. The basic measure of model accuracy used throughout this thesis is the root mean square (RMS) error between the actual (y) and predicted ($\hat{y}$) process output, and is calculated using the following relationship,

$$RMS = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)}{N}} \qquad 2\text{-}2$$

Where N is the number of data samples. This differs from Koza's approach, which measures fitness by counting the number of data points that have residual errors below a certain tolerance. This number of 'hits' increases as the prediction accuracy increases and can be used to measure the performance of each candidate solution. A major drawback of this approach is that each data point is simply classified as a 'hit' or a 'miss' and it is possible for one model to be much more inaccurate than another but still have the same fitness value. Furthermore, data from real processes may often contain noise and/or outliers, making it difficult to choose the tolerance that corresponds to a successful result.

Other error measures such as the sum of the square errors could also be used, however, RMS error has the useful property of being scaled in order to produce values that are easier to interpret. For example, if the data is scaled in the range [0 1], a RMS error of 0.01 corresponds to an average error of one percent. Although it would be convenient to use a fitness measure that increases as model performance improves, this is not the case with RMS error, which decreases as the model accuracy improves. A possible solution is to scale the RMS values using the following relationship (Hiden, 1998),

2-3

Deleted: ~~$F = \frac{1}{1+RMS}$~~

The application of equation 2-3 results in fitness values scaled in the range [0 1] with higher values corresponding to models with a lower prediction error. However, further investigation reveals that this approach may be a poor choice when combined with FPS. The disadvantages of this method and the possible alternatives are discussed in the next section.

2.3.3 Selection methods

As the selection method acts only on the fitness of each population member, the GP algorithm can use the same selection methods as the GA. One method, FPS, was outlined in section 2.2.2. Unfortunately, there are number of drawbacks associated with this technique. For example, if one population member has a particularly high level of fitness compared with the rest of the population, it will be much more likely to be selected for reproduction. If the difference in fitness is large, the probability of selection will be so great that the same individual will be repeatedly selected and the population will become dominated by a single individual. This will lead to a loss of diversity and possibly premature convergence. This could occur, for example, at the start of a GP run when the majority of the solutions perform poorly due to the probabilistic manner in which they were generated. If, by pure chance, one individual performs reasonably well, it is likely to be over-selected by FPS.

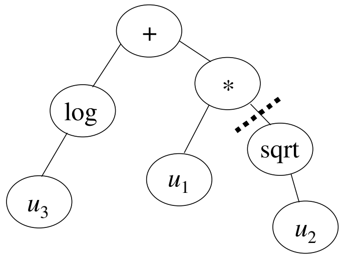
Another problem that can be encountered when using FPS occurs when the population is saturated with individuals that have very similar fitness levels. This would be especially likely to occur if equation 2-3 is used to scale certain RMS error values. For example, if the RMS errors lie in the range [0.001 0.01] (equivalent to prediction errors of between 0.1% and 1% for output data scaled in the interval [0 1]), equation 2-3 produces fitness values in the range [0.99 0.999]. The result is that there is little difference between the probability of selection for the best and worst individuals in the population and the selection process will be almost random. Because of these shortcomings, linear ranking was used throughout this work as a method of assigning fitness. This method was first introduced by Baker (1985) in order to overcome the deficiencies associated with FPS. In linear ranking, the individuals are sorted in order of their fitness. The best individual is assigned a rank of N (where N is the number of individuals) and rank 1 to the worst. A probability of selection is then assigned linearly to the individuals according to their rank.

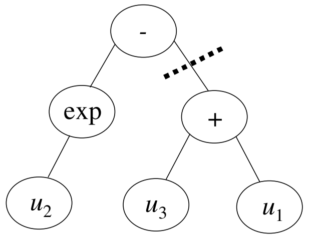
An additional reason for using linear ranking is that Chapters 6 and 7 study a multi-objective GP algorithm that employs a ranking based selection scheme. In order to compare the algorithm with the single objective version, it is necessary to use ranking for both algorithms. Other selection schemes could provide similar performance to linear ranking. For example, tournament selection is another possible technique. This involves the random selection of a group of individuals that take place in a 'tournament' with the fittest individual being selected. The process is repeated until enough individuals have been selected to generate the next population. This technique requires less processing than linear ranking, as the entire population does not have to be sorted. A detailed comparison of selection methods can be found in Blickle and Thiele (1995).

2.3.4 Application of Genetic Operators

This section describes the application of the standard genetic operators to tree structured GP model expressions. The direct reproduction operator is the same as that used by the basic GA and simply passes the chosen population member through to the next generation without alteration. The mutation and crossover operators are modified to account for the domain-specific syntax that governs the way that genetic material can be combined to produce feasible population members. For regression problems, this means that certain rules must be obeyed in order to create meaningful mathematical expressions from constants, mathematical operators and parentheses. For example, functions such as plus and minus must have two input arguments, whereas logarithm and exponential only require a single argument. The following example demonstrates how crossover is applied to two model expressions. With reference to the following parent tree structures,

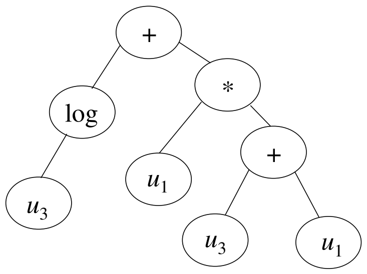
Parent 1: $\hat{y} = \log(u_3) + u_1\sqrt{u_2}$ ***Parent 2:*** $\hat{y} = \exp(u_2) - (u_3 + u_1)$

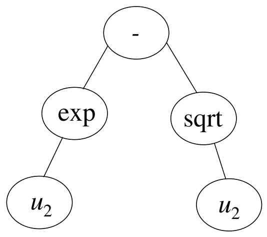




A crossover point is randomly selected on each parent (marked by a dashed line). The subtrees below these points are then exchanged to produce new individuals. In this example the ‘sqrt(u_2)’ subtree from the first parent is exchanged with the ‘u_3+u_1’ subtree from the second parent to produce the following offspring,

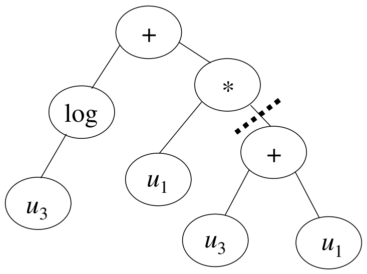
Offspring 1: $\hat{y} = \log(u_3) + u_1(u_3 + u_1)$ ***Offspring 2:*** $\hat{y} = \exp(u_2) - \sqrt{u_2}$

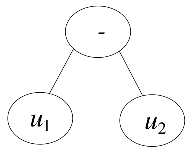




In GP the mutation operator deletes an existing subtree and replaces it with a newly generated expression. This process is illustrated using the following parent expression,

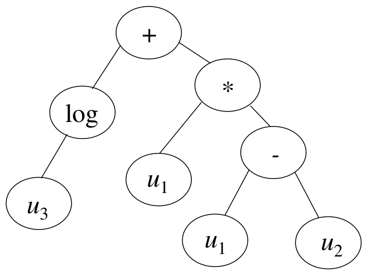
Parent: $\hat{y} = \log(u_3) + u_1(u_3 + u_1)$ ***New subtree:*** $u_1 - u_2$





A mutation point (dashed line) is randomly selected in the parent model tree and a new subtree is generated to take the place of the removed subtree.

***Offspring**:* $\hat{y} = \log(u_3) + u_1(u_1 - u_2)$



In this case, the 'u_3+u_1' subtree in the parent equation is replaced by a 'u_1-u_2' term to create a new model equation. Although the GP algorithm used in this thesis is restricted to the basic crossover and mutation operators described in this section, other operators are also possible. For example, crossover does not have to be based on two parents that have been taken from the existing population. A possible variation involves the use of one randomly generated parent tree, which is combined with a parent selected from the current population. The resulting operation has been referred to as *headless chicken* crossover and was first used in GP by Angeline (1997).

2.3.5 Algorithm control parameters

The GP algorithm requires the specification of the same basic parameters as the GA described in section 2.2, namely population size, generation gap and the probabilities of crossover, mutation and direct reproduction. Previous work concerning chemical process modelling by McKay *et al.* (1997) and Hiden (1998) has shown that population sizes as small as 50 or 100 individuals are usually sufficient to obtain accurate predictions. However, the choice of population size is clearly dependent on the problem being tackled, with some problems requiring hundreds of thousands of population members (Koza 1999). Although it is unlikely that such enormous population sizes will be required for the applications discussed in this work, it is possible that the use of larger population sizes may give more accurate model predictions and the choice of population size will be dependent on the level of accuracy required by a particular application. The population sizes selected for the runs conducted in this thesis were chosen heuristically, taking into consideration both the accuracy of model fit and the computing time available.

The correct choice of values for the probabilities of crossover and mutation has been a subject of some debate by GP researchers. Experience gained from work with GAs led many to believe that crossover was the driving force behind the evolutionary process with mutation performing a minor role. These ideas were adopted by early GP practitioners who made little or no use of mutation (for example, Koza 1992). However, as interest in the underlying mechanisms of GP increased, some researchers began to question the existing theories regarding the relative importance of crossover and mutation (Angeline, 1997). It is beyond the scope of this study to attempt to determine the optimal settings of such parameters due to the large number of runs that would be required to cover all of the possible combinations. As a result, the crossover and mutation probabilities were held constant at 0.7 and 0.2 respectively with a generation gap of 90%. These values are similar to those used in other process modelling applications (McKay *et al.*, 1997, and Willis *et al.,* 1997). Although workers such as Koza (1992) do not the use mutation or a generation gap, it was thought that their inclusion would be more beneficial in this study due to the relatively small population sizes used. In addition, Hiden (1998) performed an extensive study into the effects of using different crossover and mutation probabilities, concluding that GP appeared to be insensitive to small deviations from the values referred to above.

2.4 Conclusions

This chapter has outlined the basic underlying concepts of the GA and the possible advantages to be gained over traditional optimisation methods. The main aspects of the basic GP algorithm have been described, making particular reference to the evolution of mathematical model expressions. The next chapter outlines the modifications made to the algorithm in order to generate models of steady-state chemical processes.

3 Steady-state modelling

3.1 Introduction

The application of GP to symbolic regression problems was first demonstrated by Koza (1992). A number of problems were studied, including the modelling of polynomial systems, the discovery of trigonometric identities and econometric forecasting. Jiang and Wright (1992) described an equivalent approach that used a GA to evolve tree structured model expressions. The algorithm used a Levenberg-Marquardt routine to optimise the values of the numerical constants appearing in each model expression. Iba *et al.* (1994) described a GP algorithm called STROGANOFF (STructured Representation On Genetic Algorithms for NOn-linear Function Fitting) that was designed specifically for solving non-linear regression problems. Unlike the simple mathematical functions employed by Koza's algorithm, the functional nodes in STROGANOFF generated polynomial expansions of the inputs. The polynomial coefficients were then optimised using a regression algorithm. A possible drawback of this method is that the functional relationship is restricted and poor performance may result when the algorithm is applied to a wider range of problems.

Since then, GP has been applied to steady-state and dynamic modelling problems in a range of research areas, including chemical engineering. For example, McKay *et al.* (1997) demonstrated how GP could be used to develop steady-state models of chemical processes including a vacuum distillation column and a system containing two CSTRs arranged in series. Other engineering applications of GP to steady-state modelling include fluid systems identification (Watson and Parmee, 1996), the determination of heat flux correlations (Lee *et al.*, 1997) and the evolution of hydrometallurgical system models (Greeff and Aldrich, 1998). An interesting approach involves the use of GP in combination with linear data analysis techniques. For example, Hiden (1998) demonstrated how GP could be used in conjunction with principal components analysis (PCA) and partial least squares (PLS) to generate steady-state models of non-linear processes. The rest of this chapter demonstrates how

GP can be used to evolve models of steady-state chemical processes. A modified form of the algorithm, known as the multiple basis function (MBF) GP algorithm (Hinchliffe *et al.*, 1996, Hiden, 1998) is described and its performance compared to a more 'standard' algorithm.

3.2 Standard GP algorithm details

Symbolic regression involves the use of GP to develop a model by simultaneously evolving a functional relationship and the associated numerical parameters. In this work, the aim is to use GP to generate steady-state models of the form,

$$\hat{y} = g(\varphi, \Theta) \qquad 3\text{-}1$$

Where $\hat{y}$ is the estimated output, g is a non-linear function, φ are the regressors (in this case, the process inputs u_1, u_2,..., u_n) and Θ are the model parameters. The functional form of the approximation, g, can be generated using a GP algorithm provided with the appropriate terminal and functional sets. In this chapter, the function set (F) contains the standard mathematical operators - plus, minus, divide, multiply, power ('^'), square root ('SQRT'), square ('SQR'), natural logarithm ('LOG') and exponential ('EXP'),

$$F = \{+, -, /, *, \hat{}, \mathrm{SQRT}, \mathrm{SQR}, \mathrm{LOG}, \mathrm{EXP}\} \qquad 3\text{-}2$$

The function set is restricted to basic mathematical operators but can be extended if model performance is inadequate. Possible additions include Koza's *automatically defined functions* (Koza, 1994) and functions borrowed from other modelling techniques such as the hyperbolic tangent found in feedforward neural networks. The terminal set contains the process inputs and randomly generated numerical constants.

$$T = \{u_1, u_2, ..., u_n, \Re\} \qquad 3\text{-}3$$

The floating-point random constants ($\Re$) were generated uniformly in the range [-10 10]. Even with the inclusion of random constants, a weakness of GP is its inability to evolve numerical parameters efficiently. This is because the majority of the constants are generated during the creation of the initial population with only a small amount of new constants being produced during evolution by the mutation operator. Although the application of genetic operators on this initial set of constants can theoretically enable the algorithm to evolve any numerical value, the process is extremely inefficient.

In order to overcome this problem and maximise the performance of the evolved model structures, an optimisation routine was used to fit the numerical constants present in each model. A study of the literature reveals that a number of optimisation techniques have been used in conjunction with GP to achieve this aim. Gray *et al.* (1998) suggest that gradient-based techniques are unsuitable as they lack the robustness required to deal with the wide range of model structures produced by GP. As a result, the authors use a combination of simulated annealing (SA) and the simplex method. Cao *et al.* (1999) described an algorithm called HEMA (hybrid evolutionary modelling algorithm) that used a GA to carry out constant optimisation. Although there are benefits to be gained by using a GA, the main disadvantage is the high computational cost of applying the algorithm to every population member. The authors do not cite this as a factor in their work, but it is likely that a combined GA/GP approach would increase solution times unacceptably. A much simpler method is to use a special mutation operator to adjust the constant values. Evett and Fernandez (1998) claim that this technique is easy to implement and can improve the ability of GP to evolve numerical constants without significantly increasing computational requirements.

Bettenhausen, *et al.* (1995) argue that any conventional optimisation procedure is valid, and use a gradient descent algorithm to determine constant values. In this work, the Levenberg-Marquardt (L-M) non-linear least squares routine was used to optimise the parameter values present in each population member. The L-M algorithm (see appendix) is a commonly used non-linear optimisation routine and has been used in conjunction with GP for other modelling applications (for example, Jiang and Wright, 1992, McKay *et al.*, 1997). Hiden (1998) used a number of systems to demonstrate how a GP algorithm using L-M optimisation was able to generate models that gave more accurate predictions than those evolved without using parameter optimisation.

The important features and settings of the standard GP algorithm are summarised in Table 3-1. The algorithms developed in this thesis were all implemented using the MATLAB programming language. Although this means that execution times may be longer than would be achieved using languages such as C or FORTRAN, MATLAB's range of built-in mathematical and graphical functions lead to a significant reduction in algorithm development time. The population members were stored and manipulated in the form of variable length character strings. In order to prevent individuals from occupying excessive computational resources, a hard constraint of 500 characters was placed on the length of the models strings. Justification for the choice of selection method and the probabilities of mutation and crossover were given in section 2.3.5. From here on, this algorithm will be referred to as the 'standard GP' algorithm.

Table 3-1 – Summary of algorithm settings and parameters

Function set	+, -, /, *, ^, `SQRT`, `SQR`, `LOG`, `EXP`
Terminal set	Process inputs, $u_1,\ldots u_n$ scaled in range [0 1]
	$\Re$ generated uniformly in range [-10 10]
Crossover probability (p_c)	0.7
Mutation probability (p_m)	0.2
Direct reproduction probability (p_r)	0.1
Generation gap	90%
Parameter optimisation	Levenberg-Marquardt
Fitness measure	RMS error
Selection method	Linear ranking
Maximum tree size	500 characters

The units used to measure process variables mean that recorded values can typically vary by several orders of magnitude. To prevent the search being biased towards one particular process variable, the input-output data was scaled in the range [0 1]. It is important to note that all of the RMS error values quoted throughout this thesis are calculated on the scaled data.

3.3 Multiple Basis Function GP Algorithm

The GP algorithm described in the previous section was restricted to the development of models consisting of a single tree structure. However, some of the more established system identification methods use models constructed from a number of functions which combine to produce the overall model output. The non-linear function in equation 3-1 can be represented as a series of separate functions, often referred to as *basis* functions,

$$g(\varphi, \Theta) = \sum_{i=1}^{m} \theta_i g_i(\varphi) \qquad 3\text{-}4$$

Where m is the number of basis functions, g_i represents the individual functions and θ_i the model parameters. Ljung (1999) shows how this expansion can be used to represent and analyse virtually any non-linear modelling technique. Examples include Fourier series, wavelets and the single-layer feedforward neural network. Most well known non-linear black-box modelling techniques use multiple instances of the same basis function. Ljung refers to this as the *mother* basis function, an obvious example being the feedforward neural network, which uses a number of log-sigmoid or hyperbolic tangent functions. Another network architecture commonly used for modelling purposes is the radial basis function (RBF) network (Chen *et al.,* 1990a), which uses Gaussian basis functions. Although the modelling capabilities of such techniques have been proven over a wide range of problems, a possible weakness is that they are restricted to using a particular type of basis function. A potential advantage of GP is its ability to evolve different types of basis functions and combine them to form novel model architectures. Another disadvantage of existing techniques is that the overall model structure (for example the network architecture) is fixed before the parameter optimisation stage takes place. If the resulting model does not provide an accurate solution, more candidate structures will have to be evaluated before a successful result is obtained. An attractive aspect of the GP algorithm is that the model structure is not fixed in this way as the number of basis functions can be varied during the evolutionary process.

The incorporation of the model structure outlined by equation 3-4 within a GP framework is similar to an approach proposed by Altenberg (1994), who suggested that the limitations of a fixed GA representation could be overcome by allowing the algorithm to increase the number of genes present in each population member. This 'multi-gene' algorithm was able to expand the length of the genome in order to improve performance. Although these concepts were demonstrated using a GA, Altenberg mentions that a GP algorithm could be modified in a similar fashion. The standard GP algorithm described earlier can be easily modified to evolve models that are constructed from a number of separate functions, hopefully improving model performance by varying the number of functions as the algorithm run proceeds. Fröhlich and Hafner (1996) also described a GP algorithm that used population members made from linear combinations of separate basis functions. The algorithm used a simplex method to optimise the model parameters and was successfully applied to a number of function approximation problems. More recently, Raidl (1998) proposed a technique where standard GP trees are pre-processed in order to transform them into a linear model structure. The first stage of the transformation requires a search for the locations of the '+' and '-' nodes in each model tree. The positions of these nodes are then used to divide the parent tree into separate basis functions. The basis functions are then assigned weighting coefficients, which are optimised using a least-squares algorithm.

The MBF-GP algorithm used in this work generates models of the following form,

$$\hat{y} = \theta_o + \sum_{i=1}^{m} \theta_i g_i(\mathbf{U}) \qquad 3\text{-}5$$

Where **U** is a matrix of process inputs and θ_0 is a bias or offset term. As the model structure given by equation 3-5 is linear in the parameters, the values of $\theta_0,\ldots,\theta_m$ can be found using the method of least squares. The probabilistic nature of GP model generation may cause the standard least squares algorithm to suffer numerical problems due to ill-conditioning and/or linear correlations between basis functions. To overcome these difficulties, the Moore-Penrose pseudo-inverse (Press *et al.*, 1992) based on the method of singular value decomposition (SVD), was used to carry out the least squares optimisation. The individual basis functions were generated using the same function and terminal sets used by the standard GP algorithm. The number of basis functions used to construct the model expressions in the initial population was chosen in the range [1 10] using a uniform probability distribution.

3.3.1 Modification of Genetic Operators

Apart from crossover, which comprises two separate procedures, the MBF-GP algorithm uses the same genetic operators as the standard GP algorithm. The MBF-GP algorithm uses modified crossover operators, referred to as *high* and *low* level crossover. This technique provides a mechanism for useful material to be exchanged in the form of entire basis functions or basis function subtrees. These operators will be described with reference to the following parent expressions, each constructed from three basis functions,

Parent 1: $u_1 / u_2 + u_3 \mid 1.46 + \exp(u_1) \mid \log(u_3) + u_1$

Parent 2: $u_1 + u_2 \mid u_3 - \log(u_2) \mid 1/u_4 - u_2$

An important aspect of the algorithm implementation is that the regression parameters ($\theta_0,\ldots,\theta_m$) are not stored explicitly. These values are calculated at the fitness evaluation stage and are not affected by the genetic operators. However, as the terminal set contains real numbers, numerical constants can appear inside basis functions. These constants are not optimised, but can be subjected to crossover or mutation. With low-level crossover, crossover takes place between basis function subtrees in the conventional manner. Two possible offspring created from applying low-level crossover are as follows,

Offspring 1: $\log(u_2) + u_3 \mid 1.46 + \exp(u_1) \mid \log(u_3) + u_1$

Offspring 2: $u_1 + u_2 \mid u_3 - u_1 / u_2 \mid 1/u_4 - u_2$

Where the 'u_1/u_2' subtree from the first basis function of *parent 1* has been exchanged with the 'log(u_2)' term in the second basis function of *parent 2*. High-level crossover involves the transfer of complete basis functions between parents. For example, using the same parent models as before,

Parent 1: $u_1 / u_2 + u_3 \mid 1.46 + \exp(u_1) \mid \log(u_3) + u_1$

Parent 2: $u_1 + u_2 \mid u_3 - \log(u_2) \mid 1 / u_4 - u_2$

Two crossover points (indicated by dashed lines) are selected using a uniform probability distribution,

Parent 1: $u_1 / u_2 + u_3 \mid 1.46 + \exp(u_1) \vdots \log(u_3) + u_1$

Parent 2: $u_1 + u_2 \vdots u_3 - \log(u_2) \mid 1 / u_4 - u_2$

Basis functions are then exchanged, producing the following offspring,

***Offspring 1*:** $u_1 / u_2 + u_3 \mid 1.46 + \exp(u_1) \mid u_3 - \log(u_2) \mid 1 / u_4 - u_2$

Offspring 2: $u_1 + u_2 \mid \log(u_3) + u_1$

In this example, the third basis function in *parent 1* has been exchanged with the second and third basis functions in *parent 2*. The resulting offspring now have different numbers of basis functions, demonstrating how this mechanism enables the algorithm to adjust the number of basis functions during model evolution. This is beneficial as the number of basis functions is not restricted to a pre-specified value and the algorithm can explore more complex model structures if required.

Hiden (1998) described a MBF-GP algorithm and demonstrated how the technique was often able to generate models that gave more accurate predictions than a standard GP algorithm. The high-level crossover operator described by Hiden only allows single basis functions to be transferred between population members. In addition, the algorithm used a fixed number of basis functions to build the models in the initial population, with a modified mutation operator being responsible for increasing or decreasing the number of basis functions during evolution. The approach used here is more flexible as the high-level crossover operator allows greater increases in model size to occur. The MBF-GP algorithm uses the same crossover probability as the standard algorithm, divided equally between the high and low-level variants.

3.4 Comparison of results

The first part of this section outlines the techniques used to compare the performance of the standard and MBF-GP algorithms. The algorithms are then compared using three case studies. The first case study uses data generated by a simple test function. The other two case studies are based on process engineering systems - a vacuum distillation column and an industrial cooking extruder. The input-output data and linear models for each case study are contained in the appendix.

3.4.1 Analysis Procedure

The stochastic nature of GP means that results will vary from one run to the next. As a result, multiple runs must be performed in order to give a more accurate indication of the performance of each algorithm. In this work, sets of twenty algorithm runs were carried out for each of the systems studied. Although a greater number of runs may add greater statistical significance to the results, run time is also a consideration for evolutionary algorithms such as GP and it was thought that this number provided a reasonable compromise.

Before the sets of runs can be compared, the 'best' model from each run must be selected. Since we are interested in developing models that have the ability to generalise (fit unseen data well), an additional set of data points was used for model validation. However, models with a high level of performance on the validation data set may not necessarily meet the same standard on the training data. Also, models that fit the training data accurately may perform unacceptably on the validation data due to over-training. As a result, a compromise was found by using the sum of the RMS values over both data sets to determine the best model from each run. This model will not necessarily be taken from the final generation, as over-training may mean that these models perform poorly on the validation data. This means that the training and validation RMS errors for the whole population from every generation must be considered when choosing the best model. Once the RMS values for the best models have been collected, their distributions can be compared for different algorithms using histograms. This allows for a more detailed comparison than can be made by merely comparing statistics such as the mean, minimum and maximum error values. The validation errors for the set of 'best' models were used to construct the histograms in order to emphasise the models' ability to generalise.

Since the error distributions obtained from each set of runs may be different and not normally distributed, a non-parametric statistical test was used to compare results. In this work the Kolmogorov-Smirnov (K-S) test was use to determine whether two distributions of prediction errors were significantly different. Fonseca and Fleming (1996b) and Hiden (1998) have previously used this technique to make comparisons between different GA, GP and neural network algorithms. An advantage of this approach is that no assumptions have to be made about the distributions being compared (for example, that the distributions are normal). The K-S test is described in more detail below.

The K-S test can be formulated as a one or two-sided test. The test compares the cumulative probability distributions (F_1 and F_2) of two samples of an independent variable, x. The null hypothesis, H_0, is that the two samples are drawn from the same parent distribution. This hypothesis is tested and rejected in favour of the alternative hypothesis H_1 (the distributions are drawn from different population distributions), i.e.,

Null hypothesis: $H_0 : F_1(x) = F_2(x)$ for all x

Alternative hypothesis: $H_1 : F_1(x) \neq F_2(x)$ for some x

The first stage of the test requires the samples to be pooled to form a single array and then sorted so that the empirical distribution function, $S(x)$ can be calculated for each sample,

$S_1(x)$ = (number of observations in sample one that are less than or equal to x)/m

$S_2(x)$ = (number of observations in sample two that are less than or equal to x)/n

Where m and n are the sample sizes. If the population distributions are the same, corresponding values of the empirical distribution functions should agree reasonably well and the differences between S_1 and S_2 should be small. The K-S test statistic is defined as the maximum value, for all values of x, of the absolute values of the difference between $S_1(x)$ and $S_2(x)$,

$$D = \max\left|S_1(x) - S_2(x)\right| \tag{3-6}$$

The corresponding P-value can then be found from tables that match the product mnD to a range of confidence limits (Gibbons, 1985). A one-sided K-S test can be used in order to detect a directional shift between two samples. The null hypothesis is the same, i.e. that the distributions are taken from the same population distribution. There are two alternative hypotheses to detect shifts in different directions,

Null hypothesis: $H_0 : F_1(x) = F_2(x)$ for all x

Alternative hypotheses: $H_{1+} : F_1(x) > F_2(x)$ or $H_{1-} : F_1(x) < F_2(x)$ for some x

The one-sided K-S tests statistics are then defined as follows:

$$D_+ = \max\left[S_1(x) - S_2(x)\right] \tag{3-7}$$

$$D_- = \max\left[S_2(x) - S_1(x)\right] \tag{3-8}$$

As with the two-sided test, tables are available that compare values for mnD_+/mnD_- at a selection of confidence intervals.

Another important aspect of algorithm performance is the computational cost required to generate an acceptable solution. One method that can be used to compare the efficiency of GP algorithms is to construct performance curves (Koza, 1992). This technique was used by Hinchliffe *et al.* (1996) to demonstrate how a MBF-GP algorithm was able to develop models of the same accuracy as a standard GP algorithm by processing fewer individuals. The main disadvantage of using performance curves is that an RMS error tolerance must be specified in order to define the prediction accuracy that corresponds to a 'successful' solution. Different tolerance values may produce vastly different performance curves, making it difficult to compare algorithms objectively. Also, the technique is designed to measure GP algorithm performance and is not suitable for comparing GP with other algorithms (such as neural networks).

An alternative measure of computational effort is the number of floating-point operations (FLOPs) that have been performed. Operations such as addition, subtraction and simple function evaluations (e.g. logarithm) each represent a single FLOP. The MATLAB programming environment conveniently provides a means of estimating the number of FLOPs performed during an algorithm run. The MBF-GP and standard GP algorithms can therefore be compared using plots of the estimated number of FLOPs against the prediction error achieved. This technique will provide a fairer comparison than can be achieved by simply counting the number of individuals processed, as longer GP strings will register more FLOPs in keeping with their longer execution time. In addition, this metric is machine-independent, unlike measures such as the time taken for an algorithm run. The remainder of this section uses the techniques described above to compare the MBF and standard GP algorithms using three case studies.

3.4.2 Test system

The following non-linear test system was used to assess the performance of the two GP algorithms when applied to steady-state model development,

$$y = u_1^5 - 2u_2^3 + u_1 \qquad \text{3-9}$$

Three input vectors (u_1, u_2, u_3) consisting of 200 data points were generated as uniformly random numbers in the range [0 1]. Equation 3-9 is similar to the polynomial systems used by Koza (1992) to demonstrate the application of GP to symbolic regression problems. However, this system has more in common with a real process, as there are multiple inputs, with only two out of the three inputs having an influence on the process output. This case study will provide a demonstration of the algorithm's ability to correctly select the relevant inputs from those provided in the terminal set. Twenty runs of the MBF and standard GP algorithms were performed with a population size of 25 for 25 generations. A set of 100 data points was used for training purposes, with the remaining 100 samples being used for model validation.

A summary of the validation RMS errors from each set of runs is shown in Table 3-2 and Figure 3-1. The results show that the models generated by the MBF-GP algorithm are significantly more accurate than the standard GP algorithm. A one-sided K-S test performed at the 95% confidence level supports this conclusion.

Table 3-2 - Comparison of validation RMS values for test system

	Minimum	Mean	Maximum
MBF-GP	4.86×10^{-5}	0.0037	0.0247
Standard GP	0.0159	0.0773	0.1441

The difference is highlighted by Figure 3-1 which shows that all but one of the MBF-GP runs have generated models with validation RMS values lower than the best result obtained by the standard algorithm. The histograms also show how the MBF-GP algorithm has produced a much narrower error distribution than the standard algorithm. The model with the lowest validation RMS error evolved by the standard GP algorithm is shown below.

$$\begin{aligned}\hat{y} = (0.08113(0.1396 - 1.773u_1)^2 - 0.08113u_2(0.4652 - 0.6826u_2) \\ - 0.6772 - 0.08113u_1)^2 - 0.769u_2^2 + 0.2982u_2\end{aligned} \quad \text{3-10}$$

Equation 3-10 contains no instances of the input u_3, meaning that the algorithm has managed to successfully select the inputs required to develop an accurate prediction.

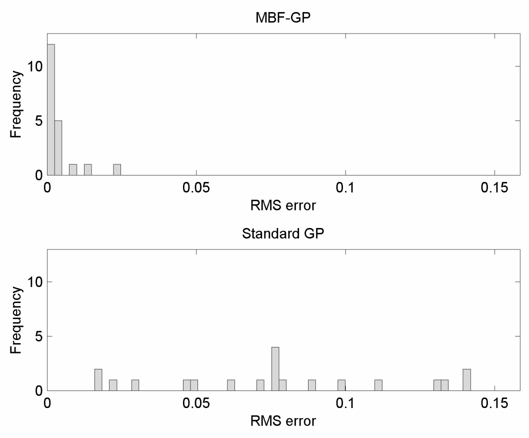


Figure 3-1 - Validation RMS distributions for test system

Table 3-3 shows the model with the best performance on the validation data evolved by the MBF-GP algorithm. The model has been displayed as a set of separate basis functions in order to give an idea of the size and form of the models developed using this algorithm. The individual basis functions have been simplified.

Table 3-3 – MBF-GP model with lowest validation RMS (test system)

Basis Function	Parameter
$u_2 - u_1$	1.615
$u_1 + 1.395^{\exp(u1)}$	2.510
0.3676^{u_2}	3.405
$1.127\exp(u_2) + 0.6352$	-0.9733
$0.2134/u_2$	-3.024×10^{-6}
$\sqrt{u_1}$	-0.04260
-0.2099	10.44
$(-1.696 + u_1)^{1/4}$	10.19
$\exp(-3.357 - 0.3404u_2)$	-558.4
$\log(2u_1)$	0.001652
$-0.3848 + u_2$	-3.734
$0.3203u_3^2$	2.042×10^{-4}
$u_1 - 3.918 - u_3$	6.788×10^{-5}
$-0.3848u_1^2$	-0.06439
$0.03878^{u_1^2}$	1.978
Bias	4.224

The model is constructed from 15 relatively simple basis functions. This demonstrates how the algorithm is able to generate population members that have a greater number of basis functions than the maximum allowed in the initial population. Unlike equation 3-10, the model shown in Table 3-3 contains some u_3 terminals (in basis functions 12 and 13). However, the model parameters associated with these terms are relatively small, which may indicate that they do not make a significant contribution towards the overall model prediction. Although it is beneficial for the algorithm to have the flexibility to adjust the number of basis function in each model, the MBF-GP model is rather complex in comparison to the actual system equation.

3.4.3 Distillation Column

This case study uses data obtained from a mechanistic model of a vacuum distillation column. The column is equipped with 48 trays, a steam-heated reboiler and a total condenser (see Figure 3-2). The control objective is to maintain the quality of the bottom product stream (x_B). While composition analysers may be used to measure product quality, investment and maintenance costs often restrict their use. Thus, a reliable inferential estimator can provide a cheap alternative to direct measurement and allow tighter control of product composition.

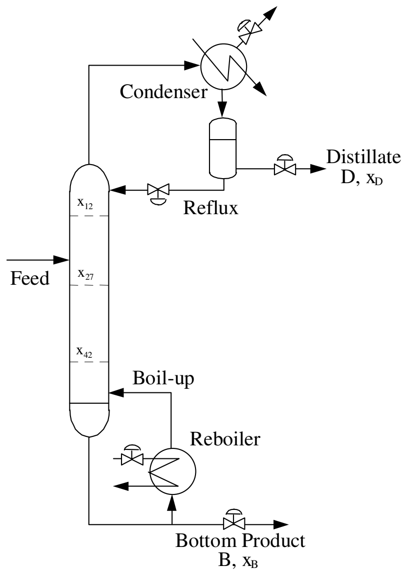


Figure 3-2- Vacuum distillation column

The column design specifications and operating data are summarised in McKay *et al.* (1997). Since the system is essentially binary, compositions can be estimated based on temperature and pressure measurements. Unfortunately, the necessary sensors are only installed on trays 12, 27 and 42 and reliable composition estimates from temperatures and pressures can only be obtained for these three trays. The objective of this exercise is to develop a model to infer the bottom product composition given the available measurements. The control system on the column ensures that it operates close to steady-state, allowing the dynamics between the tray and the product compositions to be neglected.

Input-output data was generated from a mechanistic model of the column consisting of several hundred differential and algebraic equations (Gani *et al*. 1986). One hundred and fifty steady-state composition estimates from trays 12, 27 and 42 (x_{12}, x_{27}, x_{42}) together with the corresponding values of x_B were used for model development. An additional set of 50 data records was used for model validation. Due to the increased complexity of this system in comparison to the previous case study, the number of generations and population size were increased. Twenty runs of each algorithm were performed with a population size of 50 for 50 generations. The results are summarised in Table 3-4 and Figure 3-3.

Table 3-4 - Comparison of validation RMS values (column data)

	Minimum	Mean	Maximum
MBF-GP	0.0166	0.0227	0.0341
Standard GP	0.0231	0.0329	0.0503

The results show that the MBF-GP algorithm has outperformed the standard GP algorithm on this system. The visible shift between the distributions is supported by a one sided K-S test at the 95% confidence level. As before, the errors achieved by the standard algorithm cover a wider range of values. However, the difference between the error distributions is not as exaggerated as in the previous example.

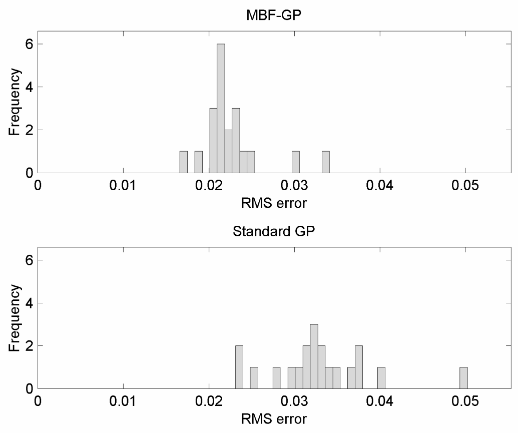


Figure 3-3 - Validation RMS distributions for distillation column data

The MBF model that produced the lowest RMS error on the validation data is shown in Table 3-5. Again, the model is rather complex, containing thirteen basis functions.

Table 3-5 – MBF-GP model with lowest validation RMS

Basis function	Parameter value
$x_{12} - \log(0.02915^{x_{27}} + 1.440)$	0.1725
$x_{12}^{x_{27}^2}$	6.335
$x_{27} + x_{42}$	0.5648
$x_{27}^{x_{27}}$	0.8651
$x_{42} - x_{42}^2 / x_{12} + 0.1799 / x_{12} - 0.5596$	-0.2511
$x_{42}^{x_{27}^2}$	1.502
$(x_{42}(x_{12} - 8.981) + 0.2932) / x_{27} - 0.5596$	-1.441×10^{-4}
$x_{27}x_{42} - x_{27}$	4.443
$(x_{12}^2)^{x_{27} - 0.5596}$	-0.1361
$\log(x_{42}) / \log(x_{27}) x_{42}$	1.798
$-3.411\log(x_{42})$	0.04516
$x_{12}^{-0.1466} + x_{12}^{4.548}$	0.02381
$x_{42}^{x_{27}}$	-3.243
Bias	-6.010

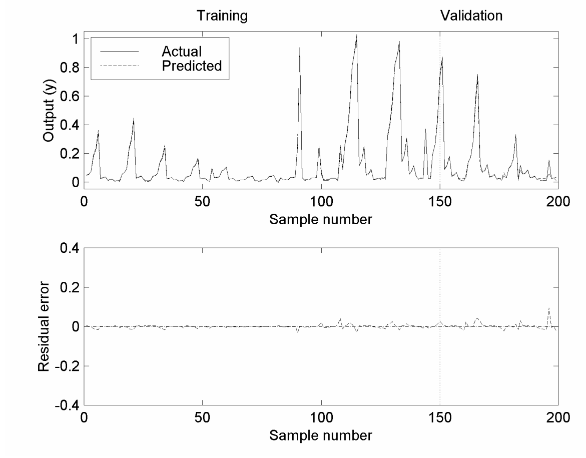


Figure 3-4 – Bottom product prediction (MBF-GP)

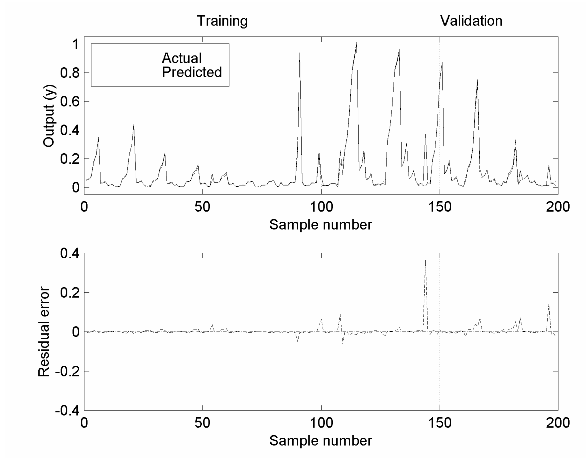


Figure 3-5 – Bottom product prediction (Standard GP)

The standard GP model that gave the best prediction for bottom product composition on the validation data is shown below in simplified form,

$$\begin{aligned} x_B = 0.132 x_{27} / \log((1.562 - x_{12} + x_{27}) / (0.388 - x_{42}) / (x_{42}(x_{12} + 1.234) - 1.045 \\ - x_{42} - x_{27}) / (1.434 - x_{42}) / (x_{42}(x_{42} - 5.643) + x_{12} - 2.199)) \end{aligned} \qquad 3\text{-}11$$

Figure 3-4 and Figure 3-5 compare the predictions obtained by the most accurate MBF and standard GP models.

Figure 3-6 compares the mean validation RMS errors achieved by the standard and MBF-GP algorithms for a given number of FLOPs. The error bars indicate +/- one standard deviation from the mean.

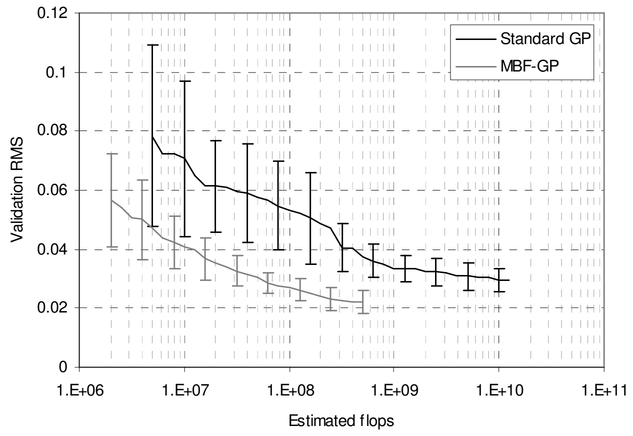


Figure 3-6 – Comparison of computational effort (distillation column)

It can be seen how the standard GP algorithm requires a greater amount of computational effort to achieve a given validation RMS error. In addition, the standard deviation error bars indicate that there is a more variation in the accuracy of fit achieved by the standard algorithm. The standard deviations are greater towards the beginning of the algorithm run in both cases. This would be anticipated as the initial population is made from a set of randomly generated model expressions.

3.4.4 Cooking Extruder

Cooking extruders have become increasingly popular in the process industry as they provide an efficient means of processing a wide range of foodstuffs. Extrusion provides a more cost efficient alternative to traditional cooking methods due to the high throughput and energy efficiency of the process. One drawback is that current knowledge of cooking extruder behaviour is largely reliant on empirical correlations and operator experience. Sensitivity to different screw and die geometries, fluid properties and complex flow patterns all combine to ensure that mechanistic models are difficult to develop. The objective of this study is to develop a model that can be used to infer the degree of gelatinisation (*g*) of the extruded product. Gelatinisation is an irreversible process that takes place when starch is heated to a sufficiently high temperature to destroy the crystalline structure of the granules. An accurate model for the degree of gelatinsation would provide a measure of the quality of the extruded starch.

A typical extruder consists of a barrel, inside which one or more helical screws rotate to convey the feed material towards a die at one end, as illustrated in Figure 3-7. The section nearest the feed point is referred to as the solids conveying zone, where the screw channels are only partially filled and there is no pressure build-up. At some point along the extruder, the channels become completely filled and the temperature and pressure increase considerably as a result of viscous heat dissipation and material transport. This section is referred to as the melt zone. Heating or cooling sections along the barrel may provide additional temperature control.

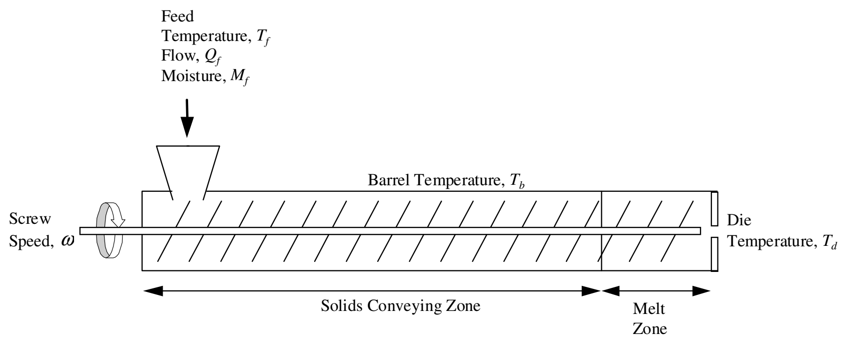


Figure 3-7 – A typical cooking extruder

The input-output data used in this study was generated using a mechanistic model of an industrial cooking extruder. The model is based on the steady-state cooking extruder model proposed by Kulshreshtha (1991) with a number of modifications and extensions made by Elsey *et al.*, (1997). The inputs available for model development were screw speed (ω), feed rate (Q_f), feed moisture content (M_f) and feed temperature (T_f). One hundred steady state records of these values along with the corresponding values of *g* were used for model development. A second set of 100 data records was used for model validation. The standard and MBF-GP algorithms were run twenty times with a population of 100 individuals for 100 generations. Table 3-6 and Figure 3-8 compare the validation RMS errors for the best models from each run.

Table 3-6 - Comparison of validation RMS values (extruder data)

	Minimum	Mean	Maximum
MBF-GP	0.0344	0.0462	0.0645
Standard GP	0.0195	0.0418	0.0762

In this case study, the most accurate model prediction on the validation data was generated using the standard GP algorithm. Figure 3-8 shows that the standard GP algorithm does not perform as consistently as the MBF-GP algorithm, producing a wider distribution of error values. Although there is an apparent dissimilarity between the error distributions, a two-sided K-S test (at the 95% confidence level) does not indicate that the difference is significant.

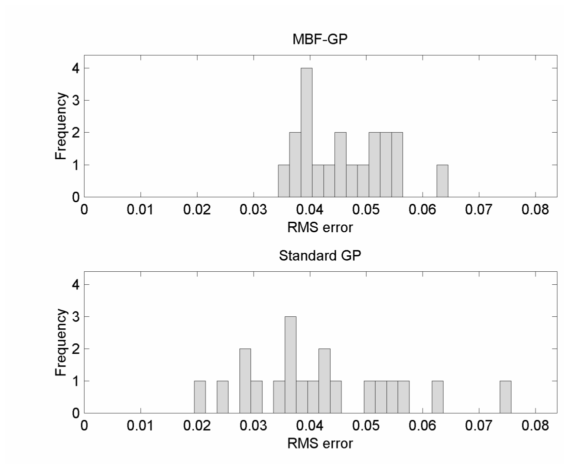


Figure 3-8 –Validation RMS distributions for extruder data

The standard GP model with the best performance on the validation data is shown below in simplified form,

$$
\begin{aligned}
\hat{g} = (&-0.7636/\exp(((-7.146-(0.1285M_f - M_f{}^2)^{1/2}(Q_f + 0.4012 - \omega \\
&+ (3.6104+\omega)\omega M_f))/\exp(0.7988\omega) + 7.596)\omega\text{^}(1.494(Q_f + M_f \\
&- \omega - 0.00696(102.3-\omega)Q_f)(3.271\sqrt{\omega} + M_f + 1.011)\omega\text{^}((0.2523M_f \\
&- M_f{}^2)^{1/2}(Q_f + 0.1365 - \omega + (0.2480+\omega)\omega M_f)(M_f + M_f{}^{3/2}))
\end{aligned}
\qquad 3\text{-}12
$$

The MBF-GP model with the lowest validation RMS error is shown in Table 3-2. The model is constructed from fifteen basis functions. Significantly, there are no instances of T_f in either model, indicating that the feed temperature is not required to develop an accurate prediction for the degree of starch gelatinsation.

Table 3-7 - MBF-GP model with best performace on extruder data

Basis Function	Parameter Value
$\omega - M_f + \log(\omega/M_f)$	0.05689
$\sqrt{M_f}$	0.3737
$6.160 - \omega$	0.5055
$\exp\left(\omega M_f{}^{\omega+1}\right)$	2.118
$M_f{}^{\omega}$	1.030
$Q_f - M_f$	1.278
ω/M_f	0.005855
$Q_f - \omega$	-1.374
$\omega^{(M_f - 0.949)}$	-4.779
7.012^{ω}	-0.06637
$0.506^{(M_f + 0.151)}$	-2.741
$M_f Q_f{}^{M_f \omega} + M_f{}^{\omega}$	-0.6438
$M_f Q_f$	0.05006
$Q_f{}^{0.07395}$	0.1486
$\left((\omega - 3.407)\log\left(Q_f\right)^{\omega}\right)^{\omega/2}$	-0.1535
Bias	0.3897

A similarity between the MBF and standard GP models is that both structures are complex and difficult to interpret. Although the models give accurate predictions, they do not provide any additional insight into the underlying physical process (apart from the exclusion of T_f). The predictions generated using these models are compared in Figure 3-9 and Figure 3-10.

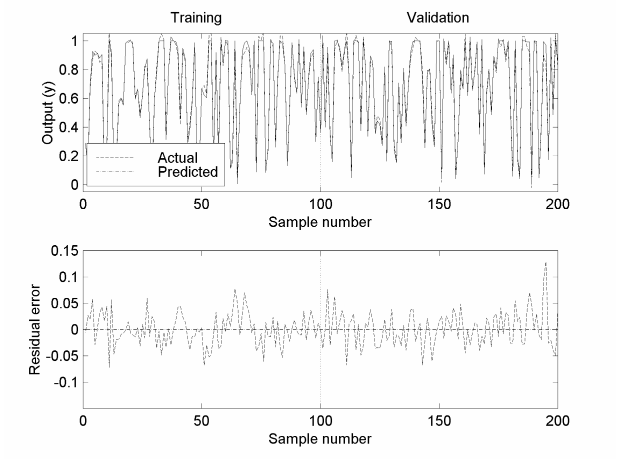


Figure 3-9 - Prediction for degree of gelatinisation (MBF-GP)

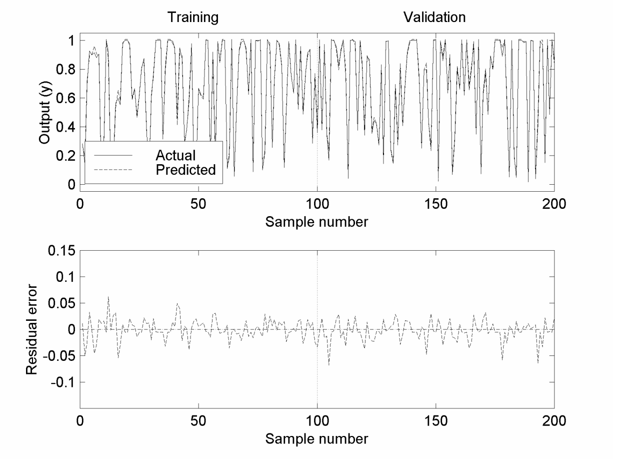


Figure 3-10 – Prediction for degree of gelatinisation (Standard GP)

Figure 3-11 shows that, as in the previous example, the standard GP algorithm requires a greater number of FLOPs in order to achieve a given validation RMS error. The difference is greater towards the beginning of the algorithm runs with the standard GP algorithm performing approximately one hundred times more FLOPs than the MBF-GP algorithm.

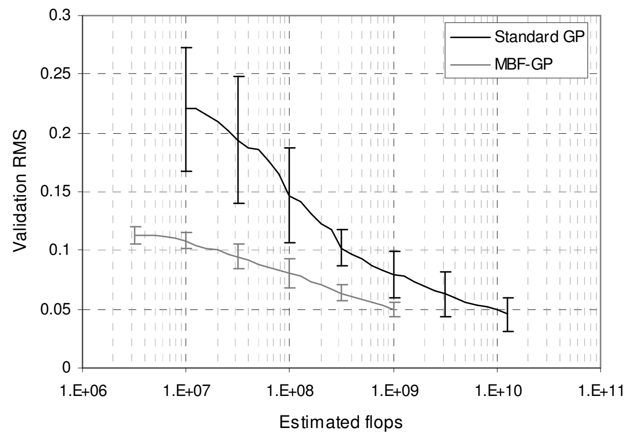


Figure 3-11 – Comparison of computational effort (extruder data)

The difference between the algorithms gradually decreases as the validation RMS error reduces. However, the computational effort is still an order of magnitude higher for the standard GP algorithm when the lowest RMS values are achieved. As was observed in the previous case study, shorter error bars emphasise the greater consistency achieved by the MBF-GP algorithm.

3.5 Conclusions

The results presented in this chapter have demonstrated the ability of both GP algorithms to generate accurate steady-state process models. The difference in model accuracy was greatest for the test system, with the MBF-GP algorithm clearly outperforming the standard GP algorithm. However, the difference was less noticeable for the two process systems, with the standard GP algorithm generating the most accurate model on the extruder data. This could be because the algorithm uses a non-linear optimisation routine to fit all of the constants in each model string whereas the MBF-GP algorithm only optimises the parameters associated with each basis function. This may enhance the standard algorithm's ability to evolve accurate models of more complex processes. The MBF-GP algorithm is at a disadvantage in the sense that it does not optimise constant values appearing inside basis functions, which could explain its tendency to construct models from a number of relatively parsimonious basis functions. In all three case studies, the MBF-GP algorithm produced narrower error distributions, indicating that it produces more consistent results than the standard algorithm.

The greatest advantage of using the MBF-GP algorithm is that it requires substantially less computational effort to achieve the same model accuracy as the standard algorithm. The L-M optimisation routine performs a relatively large number of function evaluations in order to estimate the derivatives of the prediction error with respect to each model parameter. This becomes particularly significant during the latter stages of an algorithm run when the population contains complex model strings that have a large number of numerical parameters. The least squares routine used by the MBF-GP algorithm does not have this burden and execution times are reduced considerably. This means that, although the standard algorithm was able to generate the most accurate prediction on the extruder data, MBF-GP algorithm performance could be improved by carrying out longer runs with larger populations and still require less computational effort. As mentioned earlier, a variety of optimisation techniques can be used in combination with GP, with each striking a different balance between computational complexity and the accuracy of the resulting model fit. An interesting approach would be to allow the GP algorithm to call upon an array of different optimisation methods. Each routine could be selected on a probabilistic basis, with more computationally expensive techniques having a lower chance of being employed.

The functional form of the models developed by GP has been cited as an advantage by some researchers when compared to other ‘black-box’ techniques such as neural networks. For example, Lee *et al.* (1997) found that GP was able to evolve models that performed as well as neural networks and preferred the functional form of the GP solutions. Conversely, Greeff and Aldrich (1998) suggested that GP model structures were difficult to interpret and offered no significant advantage over other methods. The work carried out in this chapter seems to support this view, as the evolved models, particularly in the case of the MBF-GP algorithm, were rather complex. Unfortunately, some of the features that are intended to improve the flexibility of the algorithm (for example, high-level crossover) may intensify this problem.

It may be possible to simplify the MBF-GP models further, perhaps by discarding basis functions that do not contribute significantly towards the accuracy of the model. The remaining functions could then be combined to form a single expression that may yield to further simplification. A more attractive approach would be for the algorithm to account for model complexity during evolution, only allowing an increase in model size to take place if an improvement in prediction accuracy is observed. The issue of model parsimony will be tackled using a multi-objective GP algorithm in chapters 6 and 7. The next chapter compares the MBF-GP algorithm with a more established data-based modelling technique - artificial neural networks.

4 Comparison of GP and Neural Networks

4.1 Introduction

The previous chapter demonstrated how GP could be used to develop accurate models of steady-state chemical processes. However, for GP to be considered as a serious alternative to more established data-based modelling techniques, the algorithm must be able to generate models of a similar accuracy without experiencing an excessive increase in computational requirements. This chapter examines this possibility by comparing the MBF-GP algorithm with artificial neural networks. An increase in low cost computing power combined with an abundance of process data has meant that the use of neural networks has increased rapidly over the last decade. Process engineering applications include fault diagnosis (Frank and Köppen-Seliger, 1997), process control (Turner *et al.*, 1996) and modelling (Lennox, 1996). Neural networks provide a means of generating accurate data-based models whilst keeping development times to a minimum. Consequently, they provide a cost effective alternative to mechanistic modelling techniques and have been applied to a wide range of steady-state and dynamic modelling problems. Another benefit is that the generic nature of neural network models means that it is relatively straightforward to incorporate them into established model–based control schemes (Henson and Seborg, 1997). This is especially relevant to the chemical industry where non-linear control methods can help to improve product quality and reduce running costs.

Previous studies (Willis *et al.*, 1997 and Hiden, 1998) have suggested that GP is capable of competing with neural networks in terms of the prediction accuracy of the evolved models. However, such comparisons tend to place little or no emphasis on the computational requirements, thus making it is difficult to draw any conclusions regarding the practicality of GP. This study aims to address this issue by comparing the performance of the two techniques in terms of the computational cost as well as the prediction accuracy achieved. The commonest network architectures used for modelling purposes are radial basis function and feedforward neural networks, otherwise known as multi-layer perceptrons. Although it may be possible to develop accurate models using either of these network architectures an assessment of both techniques is beyond the scope of this thesis. As a result, this work is restricted to the development of feedforward neural network models. The rest of this chapter introduces the fundamental concepts of feedforward neural networks and their application to steady-state modelling. Comparison is then made with GP using the case studies described in the previous chapter. Aspects of the neural network architectures and training methods used for dynamic modelling will be covered in the next chapter.

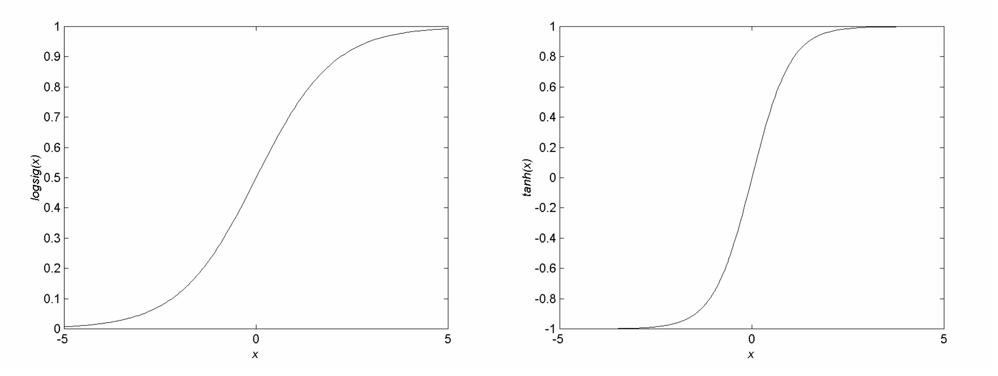
4.2 Feedforward Artificial Neural Networks

The feedforward neural network consists of a number of layers of simple processing nodes known as neurons. The output signal of each neuron is a function of the inputs to the neuron. These functions are referred to as activation or basis functions. While there is a range of possible linear and non-linear functions, the most common are the log-sigmoid and hyperbolic tangent functions, described by equations 4-1 and 4-2 respectively.

$$\text{logsig}(x) = \frac{1}{1+e^{-x}} \qquad \text{4-1}$$

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad \text{4-2}$$

Plots of these functions are shown in Figure 4-1. The output of each neuron is then passed to all of the neurons in the next network layer. Each of the connections in the network has an associated regression parameter, or *weight,* which modifies the strength of the signal that is passed along the connection to the next neuron. The adjustment of these parameters enables the network to produce the desired output value for a given set of inputs.



Log sigmoid Hyperbolic tangent

Figure 4-1 Activation functions

Figure 4-2 is a diagram of an artificial neuron, where $u_1,...,u_n$ are the n inputs, $w_1,...,w_n$ are the weights, y is the output and f is the activation function. An additional input provides a bias or offset and has an associated parameter, b. This term is analogous to the bias term used in linear regression problems and allows the activation functions to approximate a wider range of function types (Wray and Green, 1991). The summation function is sometimes referred to as the combination function of the neuron. Other types of networks, such as RBF networks use different types of combination function.

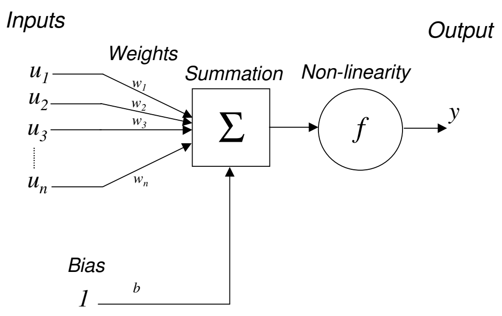


Figure 4-2 – An artificial neuron

The output of the neuron is given by equation 4-3,

$$y = f(\sum_{i=1}^{n} w_i u_i + b) \qquad 4\text{-}3$$

The outputs of each node in a network layer provide the inputs to each of the neurons in the next layer. Information flows through the network in only one direction, from the input to the output layer, hence the term *feedforward* network. The structure of a typical feedforward network containing a single hidden layer of neurons is shown in Figure 4-3. The input and output layers have linear activation functions, which means that they play no role in modelling any non-linearities associated with the input-output data. Although it is possible to use networks that have more than one hidden layer, the network architecture shown in Figure 4-3 has been successfully applied to a range of process engineering problems (for example, Willis *et al.*, 1991, Turner *et al.*, 1996 and Lennox, 1996).

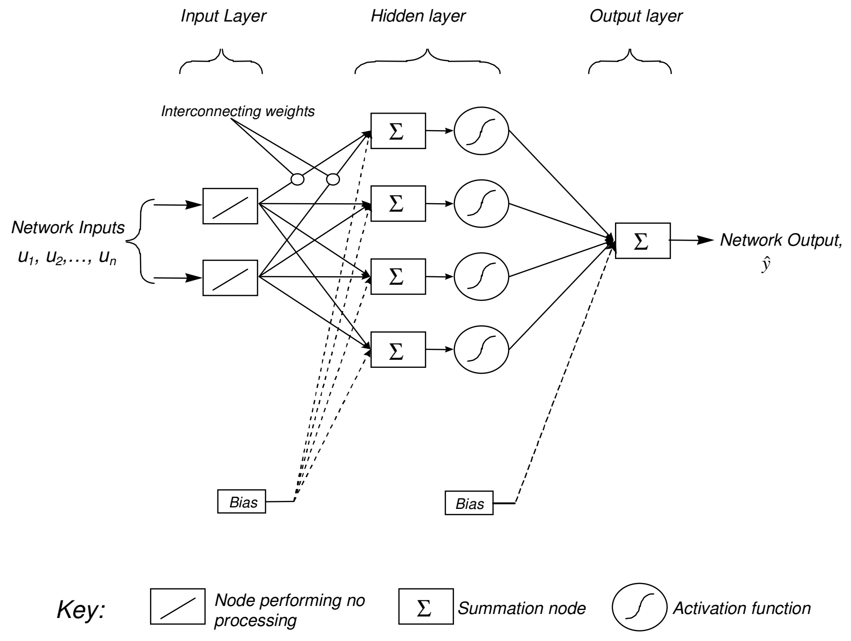


Figure 4-3 - The feedforward artificial neural network

The process of modifying the network parameters to minimise the error between the actual and predicted process output is known as network *training*. There are a number of algorithms that can be used for this purpose, the most important of which are described in the next section.

4.3 Network Training

The non-linear, highly interconnected structure of neural networks means that it is necessary to use some form of non-linear optimisation algorithm. During training, the goal is to find the values of the weights and biases that minimise the network prediction error, ε. The objective function usually takes the form of a quadratic error function, where the aim is to minimise the sum of the squared error between the predicted and actual values of the output,

$$\min_{\Theta} \varepsilon = \sum_{i=1}^{N} \left(y_i - \hat{y}_i(\varphi, \Theta) \right)^2 \qquad 4\text{-}4$$

Where Θ is a vector of network parameters, φ contains the input data samples and N is the number of data points. The method traditionally used for neural network training is known as back-propagation (Rumelhart *et al.*, 1986) and is described in the next section.

4.3.1 Back-Propagation

Back-propagation is a steepest descent algorithm that makes use of analytical gradients of the error surface. The term 'back-propagation' comes from the fact that the derivation of the gradient equations begins at the output layer and propagates back through the network. Training proceeds by calculating the partial derivatives of the prediction error with respect to each of the network weights and biases. These values can then be used to take a step in the direction of the steepest gradient. The basic parameter update rule uses a step length proportional to the magnitude of the gradient and is given by,

$$\Theta_{i+1} = \Theta_i - \alpha \mathbf{j}_i \qquad 4\text{-}5$$

Where $\mathbf{\Theta}_i$ is a vector of network parameters at iteration i, $\mathbf{j}_i$ is a vector containing the partial derivatives of the prediction error with respect to $\mathbf{\Theta}_i$ (the Jacobian), and α is the step length or *learning-rate*. In the most basic implementation of the back-propagation algorithm the learning rate remains constant, but in practice, faster convergence is achieved by changing the value from one iteration to the next.

An additional term, known as *momentum* can be included in order to increase the speed of the search. This also enables the algorithm to pass smoothly over small undulations in the error surface, making it less likely to become trapped in local minima. The parameter update equation then becomes,

$$\Theta_{i+1} = \Theta_i - \alpha \mathbf{j}_i + \eta(\Theta_i - \Theta_{i-1}) \qquad 4\text{-}6$$

Where η is the momentum term. Although back-propagation is easy to implement, a number of disadvantages must be addressed in order to achieve faster and more accurate solutions. For example, convergence can be extremely slow as the algorithm approaches a minimum point. In addition, since the algorithm always searches in the 'downhill' direction, it has no way of avoiding local minima. Because of these deficiencies, numerous variations on the standard back-propagation algorithm have been suggested. Some of the most useful of these enhancements are discussed in the next section.

4.3.2 Enhanced Back-Propagation

It is common for gradient descent algorithms to vary their step length from one iteration to the next. This is usually achieved by performing a line minimisation in order to determine the size of the optimal step length in the direction of steepest gradient. This approach has been omitted from the back-propagation algorithm due to the high degree of computational effort that is required for the additional function evaluations. An alternative method is to use an adaptive learning rate.

The basic back-propagation algorithm described above makes use of a constant learning rate, meaning that the step length is proportional to the gradient. This may lead to problems when there are large differences between the gradients of different parameter values. For example, if the learning rate is low and the gradient is very gentle, the algorithm may take a considerable length of time to reach the minimum.

Using a higher learning rate could solve this problem, but this could cause difficulties if other parameters have steep gradients. A high learning rate coupled with a steep gradient could make the algorithm take very large steps and continually step over the minimum.

One solution is to use individual learning rates for each network parameter (Jacobs, 1988). This approach leads to the following parameter update rule,

$$\Theta_{i+1} = \Theta_i - \boldsymbol{\alpha}_i^T \mathbf{Ij}_i + \eta(\Theta_i - \Theta_{i-1}) \tag{4-7}$$

Where $\boldsymbol{\alpha}_i$ is a vector of learning rates for each parameter at iteration *i*. This modified update rule enables the learning rate for each parameter to be adjusted independently. This is carried out using successive gradient values to determine whether an increase or decrease in learning rate would be beneficial. For example, if the gradient does not change sign from one iteration to the next, it can be assumed that the algorithm is making progress 'downhill' and an increase in learning rate may lead to faster convergence. However, if the sign of the gradient alternates between positive and negative values, it is likely that the algorithm is stepping over the minimum and a reduction in the learning rate is required. A simple learning rate update rule is given by,

$$\alpha_{(i+1),j} = K_{i,j}\alpha_{i,j} \tag{4-8}$$

Where $\alpha_{i,j}$ is the learning rate for the parameter *j* at iteration *i*. An adaptive learning rate update rule can then be implemented by using the following heuristic for determining the value for $K_{i,j}$,

$$\begin{aligned}
&\text{if } \frac{\partial \varepsilon_{i\text{-}1}}{\partial \theta_j}\frac{\partial \varepsilon_i}{\partial \theta_j} > 0 \quad K_{i,j} = K_1 \\
&\text{if } \frac{\partial \varepsilon_{i\text{-}1}}{\partial \theta_j}\frac{\partial \varepsilon_i}{\partial \theta_j} < 0 \quad K_{i,j} = K_2 \\
&K_{i,j} = 1 \quad \text{otherwise}
\end{aligned} \tag{4-9}$$

Where K_1 is a parameter that controls the increase of learning rate and K_2 is a parameter used to decrease learning rate. Numerous methods have been proposed to improve the performance of the standard back-propagation algorithm. For example, Minai and Williams (1990) extended Jacobs' work and suggested that individual momentum terms could be adapted in a similar way to the learning rate. The RPROP (resilient back-propagation, Riedmiller and Braun, 1993) and Quickprop (Fahlman, 1989) algorithms are other examples of training methods intended to improve convergence times.

4.3.3 Alternative training algorithms

A number of optimisation algorithms have been developed in order to overcome some of the deficiencies of steepest descent algorithms such as back-propagation. One of these is the Levenberg-Marquardt (L-M) algorithm. Hagan and Menhaj (1994) first described how the algorithm could be used to train feedforward neural networks. A disadvantage of using the L-M algorithm is that it is more computationally expensive than the back-propagation algorithm. This factor becomes increasingly important when tackling problems that require complex network structures. However, for reasonable network sizes, this drawback may be offset by the greater efficiency that the algorithm has when compared to the basic steepest descent algorithm, with convergence usually being achieved in fewer iterations. Another advantage is that the algorithm has become a widely used method of non-linear optimisation meaning that the necessary software routines are widely available. The L-M algorithm is used to train the neural networks used for dynamic modelling in chapter 5.

As mentioned previously, network training is essentially a parameter optimisation problem. This means that virtually any optimisation algorithm can potentially be used for network training. For example, the conjugate gradient algorithm is one alternative (Charalambous, 1992), and is not as computationally expensive as the L-M algorithm. A problem associated with all of the gradient-based methods discussed so far is their potential to become trapped in local minima. An approach intended to avoid this problem is the chemotaxis algorithm (Bremermann and Anderson, 1989). The algorithm adjusts the network weights by perturbing them with random values taken from a Gaussian distribution. Weight changes that lead to an improvement in network performance are accepted and the process is repeated until the convergence criteria are satisfied. Another optimisation technique that avoids the use of gradient information is the GA. The GA does not necessarily have to be used as a simple parameter optimisation algorithm and can be used to simultaneously evolve the network architecture and weights (Schaffer *et al.*, 1990). Some of the coding difficulties connected with this approach can be avoided by using a GP algorithm to evolve the network topology. For example, Esparcia-Alcázar and Sharman (1997) used a GP algorithm to evolve recurrent neural networks for signal processing. A major disadvantage of chemotaxis, GA and GP techniques is that they are more time consuming than gradient-based methods.

4.3.4 Network Parameter Initialisation

The choice of the initial values for the network parameters can strongly influence the performance of the final solution and the time required to train the network. The use of random parameter values helps to break the symmetry of the network and prevent the convergence to the same point in parameter space for every algorithm run. Care must also be taken to avoid activation function saturation, which occurs when a neuron's output is forced to one of the flat regions of the log-sigmoid or hyperbolic tangent function. This is undesirable, as the associated gradient will be very close to zero, making it difficult for gradient-based training algorithms to optimise the parameters associated with that neuron. One solution is to choose small initial values for the network parameters. However, all of the network basis functions will be grouped closely around the 'origin' of the search space. This may result in unacceptably long training times as a large amount of shifting and resizing of the basis functions will be required.

An alternative method of network initialisation, first proposed by Nguyen and Widrow (1990), ensures that the basis functions are initially spread more evenly over the input space. The principal behind this method is to suppose that each hidden layer neuron can be considered responsible for approximating the output for a small range of output values, so that the network output is actually a piece-wise linear approximation of the desired output. The subject of weight initialisation is an important area of research as the correct choice of initial parameters can vastly improve training times. For example, Yam and Chow (2000) recently proposed a new weight initialisation scheme, claimed to greatly reduce the number of training iterations required. As a detailed evaluation of these techniques is beyond the scope of this thesis, the method of Nguyen and Widrow (N-W) was adopted. McKay (1997) demonstrated how this technique resulted in more accurate network predictions when compared to random weight initialisation. In addition, training times were improved, with fewer iterations being required to achieve a given prediction error.

4.3.5 Network Parsimony and Generalisation

When developing neural network models of chemical processes the ultimate aim is to produce models that provide accurate predictions on unseen data. This raises the problem of deciding when to stop the network training process. If training is carried out for too long, the network may model the training data very accurately but be too specialised to perform well on unseen data. This phenomenon is known as over-fitting and is more likely to occur for large networks. Networks with a large number of parameters are more likely to model characteristics of the data that are not representative of the underlying process (for example, process noise).

One procedure that attempts to prevent over-fitting is known as *early stopping* (see for example, Sarle, 1995). The technique works by stopping training when the validation RMS error begins to increase. However, this approach is not valid for GP algorithms as they are constantly evolving new model structures. As a result, the validation RMS error may increase or decrease from one generation to the next and the best overall model may be produced at any stage of the algorithm run. This means that a different approach is required to find the best model generated by a GP algorithm. The same method must also be used to select the best neural network model, so that a fair comparison can be made between the two approaches. In the previous chapter, models were chosen from GP algorithm runs based on the sum of the training and validation RMS values. Consequently, the sum of these values was calculated throughout network training, with the minimum value being used to designate the 'best' neural network result.

An alternative technique designed to improve generalisation is referred to as *weight elimination*. This involves the removal of unnecessary network connections to reduce the effective number of network parameters. This is achieved by giving each network parameter the tendency to decay towards zero. The simplest way of implementing such a strategy is to modify the parameter update rule as follows

$$\Theta_{wd,i} = (1-\xi)\Theta_i \qquad \text{4-10}$$

Where $\Theta_{wd,i}$ is a vector of modified network parameters and ξ, is a small value (for example, 0.001). The drawback of this method is that all parameters are penalised when the real aim is to drive small weights to zero. This problem can be overcome by making ξ a function of the network parameters, so that only small weights are affected (Weigend *et al.,* 1991).

A number of techniques have been proposed for reducing network complexity. Details of these methods, known as pruning algorithms, can be found in Reed (1993). Another method designed to improve network generalisation is *weight decay*, (Krogh and Hertz, 1992) which works by preventing very large weight values. As a detailed assessment of these techniques is beyond the scope of this thesis, the algorithm used in this study adopts the weight elimination scheme discussed earlier.

4.4 Comparison of GP and Neural Networks on Steady-State Systems

4.4.1 Experimental procedure

The networks used in this study were restricted to a single hidden layer of neurons containing hyperbolic tangent activation functions. In order to determine the network structure that gave the best performance on each data set, batches of twenty runs were performed with networks containing a variety of hidden layer neurons (3, 5, 7, 9, 11, 13, 15). Training was carried out using an enhanced back-propagation algorithm with individual adaptive learning rates and weight elimination. Initial values for the network parameters were determined using the N-W method described in section 4.3.4. Further details of this algorithm can be found in McKay (1997). Although there are a number of possibly more efficient training algorithms available, back-propagation is an established technique and has been used quite recently by some researchers (for example, Doherty *et al.*, 1997). McKay (1997) demonstrated how networks trained using this algorithm were able to achieve the same prediction accuracy as networks trained using the L-M algorithm but required less computational effort.

The rest of this chapter uses the three case studies used in the previous chapter to compare the steady-state modelling performance of the neural network and GP algorithms. The comparison focuses on the MBF-GP algorithm due to the advantages that the algorithm has over the standard algorithm. Histograms are used to compare the validation RMS errors obtained using the algorithms and K-S tests are used to determine the significance of any observed differences. The computational complexity of each algorithm is also considered in order to determine whether GP provides a practical alternative to neural networks.

4.4.2 Test System

The results obtained using the different network architectures are summarised in Table 4-1. The results show how the validation errors decrease as the number of hidden layer neurons is increased to five neurons. Above this number, the errors begin to increase and the largest errors correspond to the most complex networks. This would be expected, as they are more likely to over-fit the training data and generalise poorly. As a result of these observations, the network with five hidden layer neurons was chosen for comparison with GP. Table 4-2 compares the performance of the neural network with the standard and MBF-GP algorithms.

Table 4-1 – Neural network validation RMS errors (test system)

Hidden layer neurons	Minimum	Mean	Maximum
1	0.0740	0.0763	0.0769
3	0.0019	0.0081	0.0582
5	0.0013	0.0036	0.0084
7	0.0017	0.0039	0.0071
9	0.0017	0.0039	0.0071
11	0.0023	0.0042	0.0088
13	0.0023	0.0047	0.0089
15	0.0019	0.0047	0.0074

Table 4-2 - Comparison of validation RMS errors (test system)

	Standard GP	MBF-GP	Neural Network (3-5-1)
Minimum	0.0159	4.86x10-5	0.0013
Mean	0.0773	0.0037	0.0036
Maximum	0.1441	0.0247	0.0084

Figure 4-4 compares the distributions of the validation RMS errors achieved using the neural network and MBF-GP algorithms, and shows that the GP algorithm generated the most accurate models for this system.

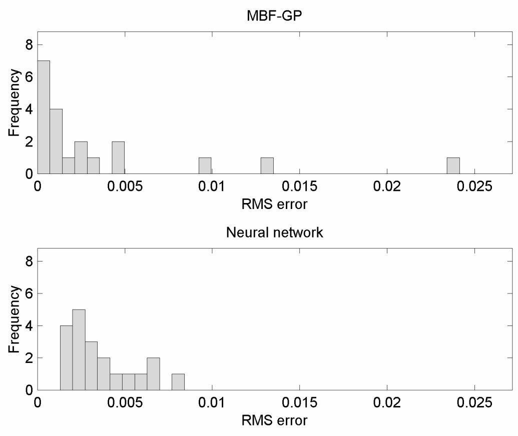


Figure 4-4- Comparison of validation RMS distributions for test system

It is possible that the GP algorithm is at an advantage in this case as its function set contains the mathematical operators that make up the system equation, whereas the neural network must approximate the function using hyperbolic tangents. A shortcoming of the GP algorithm is that it generated a handful of models with relatively poor RMS errors. The neural network does not suffer from this problem and is the more consistent of the two algorithms. A two-sided K-S test performed at the 95% confidence level indicates that the difference between the two distributions is significant.

4.4.3 Distillation Column

The network architectures used in the previous section were applied to the distillation column data. The performance of the various networks on the validation data is summarised in Table 4-3.

Table 4-3 – Summary of neural network validation RMS errors (distillation column)

Hidden layer neurons	Minimum	Mean	Maximum
1	0.0362	0.0775	0.1051
3	0.0166	0.0463	0.0900
5	0.0147	0.0318	0.0507
7	0.0147	0.0302	0.0565
9	0.0148	0.0291	0.0411
11	0.0143	0.0285	0.0523
13	0.0162	0.0292	0.0580
15	0.0190	0.0293	0.0511

The network containing eleven hidden layer neurons produced the lowest mean and minimum validation RMS. This distribution of validation RMS errors achieved using this network is compared to the distribution obtained by the MBF-GP algorithm in Figure 4-5. In this example, the GP algorithm has produced the narrower distribution, indicating that the algorithm's performance is more consistent when applied to the validation data set. The neural network is less consistent but was able to develop the most accurate model overall.

Table 4-4 - Comparison of validation RMS errors (distillation column)

	Standard GP	MBF-GP	Neural Network (3-11-1)
Minimum	0.0231	0.0166	0.0143
Mean	0.0329	0.0227	0.0285
Maximum	0.0503	0.0341	0.0523

A two-sided K-S test at the 95% confidence level verifies that the two error distributions are significantly different.

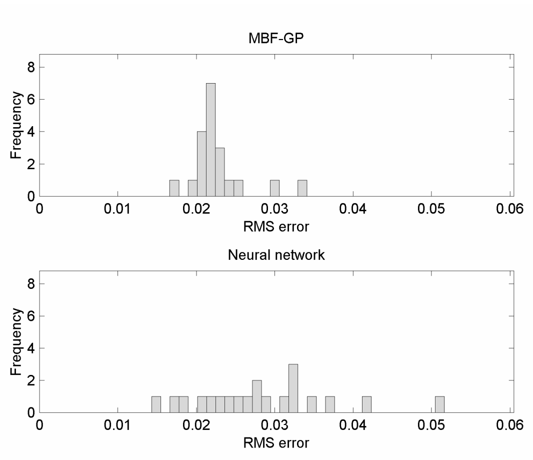


Figure 4-5 - Validation RMS distributions for distillation column data

The fixed architecture used by the neural network would perhaps be expected to produce a narrower error distribution than the GP algorithm, which explores a wide range of model structures. A possible explanation is that the neural network has a relatively large number of parameters, which means that the training algorithm is presented with a complex error surface. This will increase the number of local minima in which the training algorithm may become trapped.

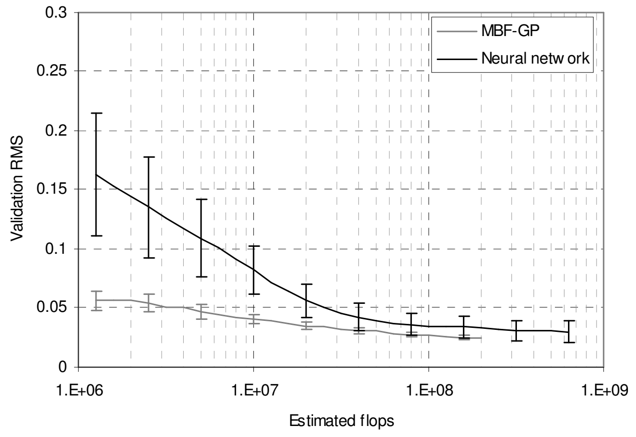


Figure 4-6 – Comparison of computational effort for distillation column

Figure 4-6 shows that the initial neural network prediction error is substantially higher than that of the GP algorithm. This could be due to the probabilistic manner in which the initial weights are chosen. Although the MBF-GP algorithm uses an initial population of randomly generated model structures, the model parameters are optimised in order to achieve the best possible prediction. This process is relatively economical in terms of computational effort and allows the algorithm to obtain a reasonably good fit early in the algorithm run. As the number of FLOPs increases, the neural network errors decrease more rapidly than the MBF-GP algorithm and the difference between the algorithms becomes less significant. The network has a relatively large number of parameters, which may explain why the GP algorithm can compete in terms of computational effort. It is also likely that the difference in algorithm performance is system dependent. The function set supplied to the GP algorithm may be suited to developing accurate models of this particular system, but may lead to inferior performance on other systems.

4.4.4 Cooking Extruder

The validation RMS error values obtained by various neural network architectures are summarised in Table 4-5. The network containing seven hidden layer neurons was selected for comparison with GP, as the mean validation RMS is the lowest for this network.

Table 4-5 – Summary of neural network validation RMS errors (cooking extruder)

Hidden layer neurons	Minimum	Mean	Maximum
3	0.0177	0.0225	0.0273
5	0.0119	0.0192	0.0278
7	0.0128	0.0186	0.0275
9	0.0128	0.0197	0.0263
11	0.0168	0.0208	0.0280
13	0.0153	0.0201	0.0245
15	0.0157	0.0216	0.0302

The neural network results were initially compared with a GP algorithm using a population size of 100 individuals run for 100 generations. The model errors are compared in Table 4-6 and Figure 4-7. The results show how the neural network produced predictions of a significantly higher accuracy than those evolved by the GP algorithm (this observation is supported by a one-sided K-S test at the 95% confidence level). Hiden (1998) reported similar results for this system, suggesting that a sigmoidal relationship in the data set meant that neural networks are ideally suited to modelling this system.

Table 4-6 - Comparison of validation RMS values (cooking extruder)

	Neural network (4-7-1)	MBF-GP (M=100, G=100)	MBF-GP (M=500, G=500)	MBF-GP + tanh (M=500, G=500)
Minimum	0.0128	0.0344	0.0192	0.0096
Mean	0.0186	0.0462	0.0293	0.0239
Maximum	0.0275	0.0645	0.0434	0.0359

To discover if it was possible for the MBF-GP algorithm to achieve the same level of accuracy as the neural network, an additional set of twenty runs was performed with a population size of 500 individuals for 500 generations. To investigate whether the neural network’s use of hyperbolic tangent functions was a contributing factor, another set of runs was carried out with this function added to the GP algorithm’s function set. The results of the additional runs are included in Table 4-6 and Figure 4-7.

Figure 4-7 shows how the validation RMS errors are reduced by increasing the population size and number of generations (a one-sided K-S test at the 95% confidence level indicates that the difference between the distributions is significant). Although the inclusion of the hyperbolic tangent function appears to have further improved model performance, a one-sided K-S test reveals that the improvement is only significant at the 90% confidence level. The resulting error distribution covers a wider range of values than the neural network distribution. The most accurate model was evolved by the GP algorithm, with a validation RMS of 0.0096 compared to 0.0128 for the neural network.

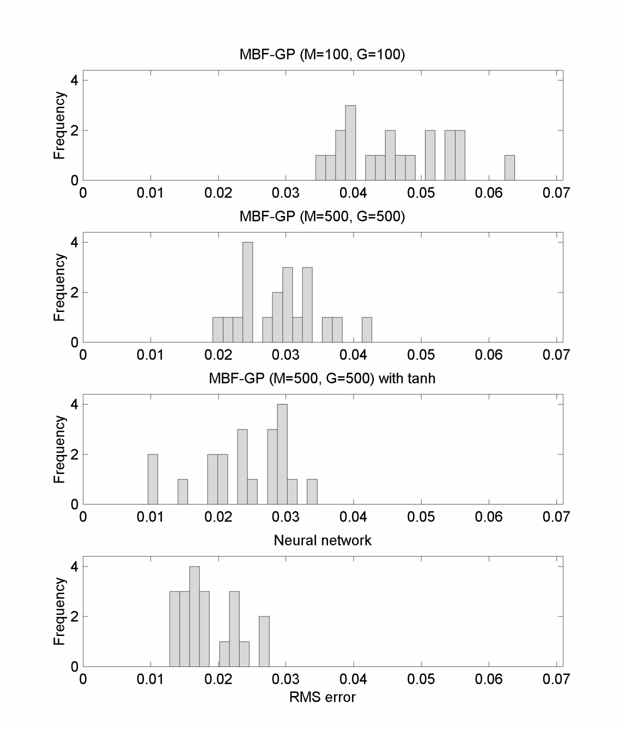


Figure 4-7 – Comparison of validation RMS distributions (extruder data)

Although the MBF-GP algorithm was eventually able to match the performance of the neural network in terms of the prediction accuracy, the resulting increase in computational effort is extremely large.

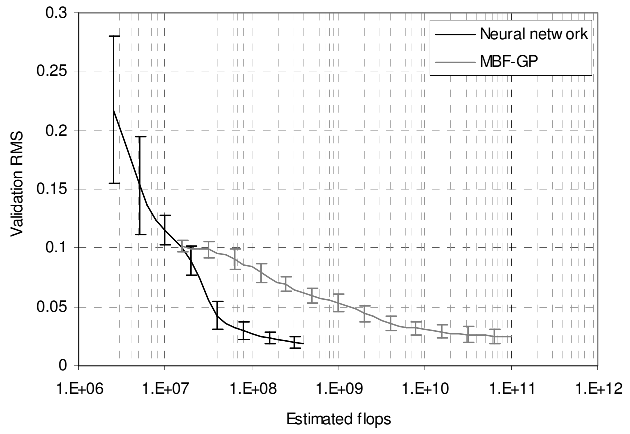


Figure 4-8 – Comparison of computational effort required for extruder modelling

Figure 4-8 demonstrates how the MBF-GP algorithm requires substantial increases in processing power to evolve models with progressively lower validation RMS values. When the lowest RMS errors are achieved, the computational cost of using the GP algorithm is more than two orders of magnitude higher than required for the neural network.

4.5 Conclusions

The work carried out in this chapter has shown that neither algorithm consistently outperformed the other, with the relative performance of the two techniques appearing to vary from one case study to the next. The GP algorithm was able to generate the most accurate models for the test system and had a slight advantage in terms of computational cost on the distillation column data. The neural network produced more compact error distributions for the test system and extruder case study, but was less consistent on the distillation column. The biggest difference was seen on the cooking extruder data, where a large increase in computational effort was required to enable GP to match the accuracy of the neural network models.

It is difficult to make fair comparisons between the GP and neural network algorithms for a number of reasons. Firstly, the computational cost profiles only compare the 'best' neural network architecture with the MBF-GP algorithm. This does not account for the fact that batches of runs must be carried out in order to determine this architecture. This can be rather time consuming, as a range of network structures must be considered. Advocates of both techniques would argue that the efficiency of each algorithm could be improved. For example, algorithms such as L-M or conjugate gradients may reduce network training times in certain cases. GP algorithm performance could be enhanced by fine-tuning control parameters or using a different function set. The performance of both algorithms may have improved if redundant process variables had been omitted from the input data sets. Although the case studies suggest that GP is able to automatically select the relevant input variables, a more detailed study is required to fully assess how performance is affected by varying degrees of redundancy in the input data.

It is also important to remember that FLOP counts are only an estimate of the computational burden. This is particularly relevant in the case of the GP algorithms, which store model expressions in the form of character strings. As string operations do not register as FLOPs, the counts do not measure the processing required to generate the initial populations of individuals or apply the crossover and mutation operators. Fortunately, these operations do not account for a significant proportion of the overall processing requirements. The most computationally expensive part of the algorithm is the fitness evaluation stage, which is responsible for more than 95% of the total processing time.

The main conclusion to be drawn from this study is that the modifications made to the standard GP algorithm have resulted in a modelling technique that is more competitive with neural networks. In this respect, the results presented in this chapter provide sufficient motivation for continued research into the benefits of using GP as a model development tool. The next section builds on the work presented so far and describes how the standard and MBF-GP algorithms can be used to evolve models of dynamic systems.

5 Dynamic Modelling

5.1 Introduction

In chapter 3 it was demonstrated how GP can be used generate accurate steady-state models of chemical processes. It was shown how a MBF-GP algorithm could be used to develop steady-state models and comparison was made with a ‘standard’ GP algorithm. Although steady-state models are sometimes useful for solving process engineering problems, applications such as process simulation and control require a model that accurately describes the dynamic behaviour of the system.

For steady state systems, the process output is uncorrelated with time and individual data records are independent of each other. This is not the case with a dynamic system, where the process output is related to previous values of the inputs and/or outputs. Consequently, the GP algorithm used for steady-state modelling cannot be applied to dynamic processes without alteration. This chapter describes how GP can be modified in order to evolve models of dynamic systems. It is shown how the MBF-GP algorithm can be applied to dynamic model development and its performance compared with the standard GP algorithm.

5.2 Modelling Process Dynamics using GP

Although it is possible to model dynamic systems mechanistically, the process can be difficult and time consuming due to the large number of equations that may be required to describe the system. In addition, if some of the chemical or physical processes are poorly understood, the resulting model may be prone to inaccuracies. A possible solution is to use a GP algorithm to automatically generate and evolve the necessary differential and algebraic equations. This method was used by Gray *et al.* (1998) to evolve expressions that were incorporated into a set of ordinary differential equations (ODEs) representing the flow of water in a coupled tank system. The authors suggested that this approach may yield a more ‘meaningful physical model’ than using a discrete time approximation. Similarly, Cao *et al.* (1999) used a GP algorithm to evolve sets of ODEs in order to model the kinetics of chemical reactions. The main disadvantage of this method is that a set of ODEs must be integrated in order to evaluate the fitness of each population member. This will be computationally expensive, especially for systems that require more than a couple of ODEs. It was shown in chapter 3 that, even for a simple test case, GP is unable to generate an exact representation of the underlying system. This is not unexpected, as GP evolves models probabilistically, meaning that the final solution is always likely to approximate the actual system. Although the GP derived model may give good predictions, its structural form may be relatively complex and difficult to interpret. It follows that it is unlikely that GP will be able to evolve ODEs that provide any insight into the underlying physical processes of a dynamic system. Consequently, there are no benefits to offset the potentially large computational cost of using GP within an ODE framework.

A method often used by process engineers is to apply the Laplace operator (*s*) to represent the problem in the *s*-domain. This approach is convenient as ODEs in the time domain are transformed into linear equations in the s-domain. For example, a first order ODE is transformed into a first order transfer function. One of the simplest approaches used by process engineers is to use a first-order plus dead-time transfer function model. For a single input-single output (SISO) system, the estimated process output ($\hat{y}$) is given by,

$$\hat{y}(s) = \frac{K_p e^{-t_d s}}{\tau_p s + 1} u(s) \qquad 5\text{-}1$$

Where, u is the input, s is the Laplace operator, τ_p is the process time constant, K_p is the process gain and t_d is a time delay term. McKay *et al.* (1996) described a technique for dynamic model development using a combination of first-order transfer functions and GP. The first step involved the fitting of a dynamic model of the form,

$$\hat{y} = \sum_{i=1}^{n} \frac{K_{p,i} u_i}{\tau_{p,i} s + 1} \qquad 5\text{-}2$$

Where $u_1,\ldots,u_n$ are the process inputs. The GP algorithm is then used to develop a model of the residuals produced by equation 5-2 instead of the original data. A disadvantage of this technique is that the dynamic characteristics of the process are fixed before the evolutionary stage of model development takes place. Another drawback is that the method assumes a first order relationship between the model input(s) and output. This assumption may be adequate for some applications, but it is unlikely to be suitable for a wider range of chemical process systems, which tend to be highly non-linear. The methodology does not include a means of compensating for process dead time, so any substantial time delays will have to be identified and removed from the data before the algorithm is applied.

Hiden (1998) improved on this approach by using a MBF-GP algorithm to build models containing first order transfer functions. This resulted in the following model form,

$$\hat{y} = a_0 + \sum_{i=1}^{m} a_i \frac{g_i(\mathbf{U})}{\tau_i s + 1} \qquad 5\text{-}3$$

Where τ_i is the time constant associated with the i^{th} basis function, $a_0,\ldots,a_m$ are model parameters, $g_1,\ldots,g_m$ are basis functions determined by the GP algorithm and **U** is a matrix containing the inputs $u_1,\ldots,u_n$. The parameters $a_0,\ldots,a_m$ were found using the method of least squares, while the values for the time constants, τ_i, were found by using a modified mutation operator. This approach has several advantages over that outlined by McKay *et al.* (1996). Firstly, the time constants are modified as the evolutionary process proceeds, instead of being fixed by a separate modelling step. In addition, the GP algorithm is now responsible for the development of the dynamic model terms. This means that the model can be constructed from non-linear expressions, giving it the potential to describe more complex systems. Finally, each basis function can be a function of more than one input enabling the model to account for interactions between process variables. Despite these improvements, Hiden's approach still relies on the use of first order transfer functions and does not provide a method for time delay identification. If the aim is to develop an automated model building tool, the GP algorithm must be able to identify the relevant process time delays and higher order dynamics.

Gray *et al.* (1998) also used GP to develop transfer function models, extending the methodology to include higher order transfer functions and time delay terms. The models were represented in block diagram form using the SIMULINK (Checkoway and Kirk, 1992) toolbox for MATLAB. In a similar approach, Bettenhausen *et al.* (1995) used a GP algorithm with a function set containing feedback loops and recursive nodes to build models of a biotechnology process, also in block diagram form. Although this approach proved successful in terms of model accuracy, a major disadvantage was that the procedure was computationally intensive, requiring a network of workstations to achieve a solution within an acceptable timeframe. This is also likely to be a problem with the method outlined by Gray *et al.* as a separate SIMULINK block diagram will have to be executed to calculate the fitness of every population member.

An alternative method, used for modelling dynamic processes is to use a time series approach, where the output is modelled as a function of past values of the input(s) and output,

$$\hat{y}_k = f(y_{k-1},...,y_{k-\tau},u_{1,k-1},...,u_{1,k-\tau},...,u_{n,k-1},...,u_{n,k-\tau}) \qquad \text{5-4}$$

Where k is the current time and τ is the maximum time shift. Introducing the back-shift operator, q^{-1} (for example, $q^{-1}y_k=y_{k-1}$), equation 5-4 may be written,

$$\hat{y}_k = f(y_k,u_{1,k},...,u_{n,k},q^{-1}) \qquad \text{5-5}$$

The function f can easily be developed using a GP algorithm with a terminal set containing lagged values of the input and output variables. The simplest form of the time series model is to predict the output based solely on past values of the input(s). This is sometimes referred to as the finite impulse response (FIR) model (Söderström and Stoica, 1989),

$$\hat{y}_k = f(u_{1,k-1},...,u_{1,k-\tau},...,u_{n,k-1},...,u_{n,k-\tau}) \qquad \text{5-6}$$

A disadvantage of this model form is that it may be necessary to use a large time history of process inputs to accurately capture the dynamic characteristics of the process. This may be especially problematic when using a GP algorithm, as the terminal set would have to be very large. This would increase the likelihood of redundant information being present in the terminal set, leading to a degradation of algorithm performance. An increase in model parsimony can be achieved by using past output values of the process. A common linear example of equation 5-4 is the Auto-Regressive Moving Average with eXogenous inputs (ARMAX) model (Söderström and Stoica, 1989). For a SISO system,

$$y_k + a_1 y_{k-1} + ... + a_{n_a} y_{k-n_a} = b_1 u_{k-1} + ... + b_{n_b} u_{k-n_b} + e_k + c_1 e_{k-1} + ... + c_{n_c} e_{k-n_c} \qquad 5\text{-}7$$

Or in simplified form,

$$A(q^{-1})y_k = B(q^{-1})u_k + C(q^{-1})e_k \qquad 5\text{-}8$$

Where A, B, and C are polynomials in the back-shift operator and e_k is a noise term,

$$\begin{aligned} A(q^{-1}) &= 1 + a_1 q^{-1} + ... + a_{n_a} q^{-n_a} \\ B(q^{-1}) &= b_1 q^{-1} + ... + b_{n_b} q^{-n_b} \\ C(q^{-1}) &= 1 + c_1 q^{-1} + ... + c_{n_c} q^{-n_c} \end{aligned}$$

a, b and *c* are sets of model parameters that must be determined and n_a, n_b and n_c are the maximum time-shifts for *y*, *u* and *e* respectively. In reality, the noise term is not measurable and is not available for model development.

The non-linear form of this model is known as the polynomial NARMAX (Non-linear ARMAX) model (Chen and Billings, 1989) and consists of polynomials made from linear and non-linear combinations of past values of *y*, *u* and *e*. The polynomial NARMAX model may be written as follows.

$$y_k = a_0 + \sum_{i_1}^{n} a_{i_1} x_{i_1} + \sum_{i_1=1}^{n} \sum_{i_2=i_1}^{n} a_{i_1 i_2} x_{i_1} x_{i_2} + ... + \sum_{i_1=1}^{n} ... \sum_{i_l=i_{l-1}}^{n} a_{i_1} ..._{i_l} x_{i_1} ... x_{i_l} + e_k \qquad 5\text{-}9$$

Where a_i are the model parameters, *n* is the sum of the maximum number of lags for *y*, *u* and *e, x_i* are the lagged terms in *y*, *u* and *e*, and *l* is the degree of the polynomial.

If no terms containing e are used, the model becomes a NARX (Non-linear Auto-Regressive with eXongenous inputs) model. For example, for a SISO system, a NARX model limited to a second order polynomial and a single process lag for the input and output is as follows,

$$\begin{aligned} y_k &= a_0 + a_1 u_{k-1}^2 + a_2 u_{k-1} + a_3 y_{k-1}^2 + a_4 y_{k-1} + a_5 u_{k-1}^2 y_{k-1}^2 \\ &\quad + a_6 u_{k-1}^2 y_{k-1} + a_7 u_{k-1} y_{k-1}^2 + a_8 u_{k-1} y_{k-1} \end{aligned} \qquad 5\text{-}10$$

Although this model structure has been used in a number of practical applications, one of the difficulties associated with NARMAX model development is that the number of possible model structures can be very large. This will be especially true for non-linear chemical processes that may have multiple inputs and require higher order polynomials with a large time history of input(s) and outputs. It will be difficult to select the appropriate terms from all of the possible combinations described by equation 5-9 as there may be hundreds or even thousands of candidate models from which to choose.

This problem can be addressed by applying techniques such as *forward* and *backward* regression, which allow model terms to be added or removed systematically. In forward regression, (Draper and Smith, 1981) model terms are added one at a time, based on their degree of correlation with the process output. This method does not account for the fact that newly added model terms may render some of the existing terms unnecessary. Consequently, the resulting model will not necessarily provide the most parsimonious solution to the problem. Backward regression (Smillie, 1966, Draper and Smith, 1981) initially estimates the parameters for a model containing all of the possible model terms. Terms are then removed and the parameters are re-estimated in order to eliminate unnecessary terms. One of the problems of this approach is that the initial parameter estimates for the model containing all of the possible terms may be ill-conditioned and produce inaccurate estimates. More efficient methods have been proposed for NARMAX model development, for example, Billings and Voon (1986a) described a stepwise regression algorithm, which uses a combination of forward and backward regression methods. More recently, Mao and Billings (1997) developed the minimal model structure detection (MMSD) algorithm, which uses a GA to determine the order in which terms are added to the overall model.

An alternative to these methods is to use a GP algorithm to determine the necessary combinations of inputs, outputs and process lags. GP has been used to develop time series models in a number of research areas. Applications include the prediction of financial markets (Chen and Yeh, 1997, Kaboudan, 1999), sunspot prediction (Jaske, 1996), modelling of hydrological systems (Babovic, 1998) and the identification of chaotic time systems (Howard and Oakley, 1995). Rodríguez-Vázquez and Fleming (1998) applied GP to the development of NARMAX models for gas turbine engine identification. More recently, Kulkarni *et al.* (1999) used GP to develop ARX models of industrial processes including a CSTR and a heat exchanger. This form of model structure is attractive in terms of its implementation within a GP framework, as few modifications have to be made to the steady-state algorithm. In addition, solution times should be more manageable than those achieved using methods based on block-diagram representations or that require sets of differential equations to be solved. The next section outlines the modifications that must be made to the standard GP algorithm to allow the development of models of this form.

5.3 Dynamic GP Algorithm Details

The standard GP algorithm is almost identical to the algorithm used for steady-state model development in chapter 3. The main difference is that modified terminal and function sets must be used to allow the algorithm to generate models in time series form. The method used to represent this type of model structure is outlined in the next section.

5.3.1 Dynamic Model Representation

In order to develop models that can be used for long-term prediction, we must assume that the actual process output values $(y_{k-1},..., y_{k-\tau})$ are unknown and cannot be used as model inputs. The predicted output values $(\hat{y}_{k-1},...,\hat{y}_{k-\tau})$ generated by the model must therefore be used instead. The general model form given by equation 5-5 then becomes,

$$\hat{y}_k = f(\hat{y}_k, u_{1,k},...,u_{n,k}, q^{-1}) \qquad 5\text{-}11$$

This form of prediction is sometimes referred to as 'pure' prediction (Henson and Seborg, 1997) as the method only requires the process inputs in order to predict the output over the entire data set. This is especially important if the model is to be used for carrying out process simulations, as the output must be assumed unknown. In addition, if GP is to be developed into a useful dynamic modelling tool it must be able to produce models that have the same level of performance and range of application as existing artificial intelligence techniques. Since established techniques such as globally and locally recurrent artificial neural networks are able to generate long-term predictions, it follows that it would be more appropriate to apply GP to the same problem. The simplest way to enable GP to construct dynamic models of the form shown by equation 5-11 is to provide a terminal set that consists of time shifted process input(s) and the model output,

$$T = \left\{ \hat{y}_{k-1},...,\hat{y}_{k-\tau_y}, u_{1,k-1},...,u_{1,k-\tau_u},...,u_{n,k-1},...,u_{n,k-\tau_u} \right\} \qquad 5\text{-}12$$

Where n is the number of process inputs and τ_u and τ_y are the maximum time-shifts for the inputs and output respectively. An important aspect of this method is that the maximum number of process lags must be chosen before the algorithm run. If this number is chosen incorrectly, the algorithm will be unable to evolve accurate models and additional sets of runs will have to be performed. Although this problem will arise if τ is less than the required value, poor performance could also result if the value is too high, as the algorithm may have to work with a large number of superfluous terminals. It is possible that the input-output data could be analysed before any runs were undertaken in order to estimate process time constants and time delays, but this would mean that GP was no longer operating as an automated modelling tool with no *a priori* assumptions.

A more elegant solution is to provide the algorithm with building blocks that allow the number of process lags to be adjusted as the run proceeds. This can be accomplished by including the back-shift operator, q^{-1}, in the function set. Dynamic models can then be created from a smaller terminal set consisting solely of the process input(s) and model output shifted by a single time sample, i.e.,

$$T = \{u_{1,k-1}, \dots, u_{n,k-1}, \hat{y}_{k-1}\} \qquad \text{5-13}$$

For example, consider the following model equation,

$$\hat{y}_k = \hat{y}_{k-2} + u_{k-3} + u_{k-4} \qquad \text{5-14}$$

The model described by equation 5-14 can also be represented by applying the appropriate back-shift operators to u_{k-1} and y_{k-1} terms,

$$\hat{y}_k = q^{-1}\hat{y}_{k-1} + q^{-2}u_{k-1} + q^{-3}u_{k-1} \qquad \text{5-15}$$

It should be noted that equation 5-15 makes use of back-shift operators that shift the terminals back by multiple time samples, for example q^{-3} shifts u_{k-1} by three samples resulting in a u_{k-4} term. The same result could be achieved by repeatedly applying single sample back-shifts but it would be impractical for the algorithm to identify large process time delays in this manner. Consequently, it is necessary to specify a maximum number of time samples (τ_{max}) for the back-shift operators in the function set. This approach is extremely flexible as the crossover operator allows the GP algorithm to evolve models that contain time-shifts of greater than τ_{max} samples. This is illustrated by the following example.

Figure 5-1 depicts two dynamic model trees constructed from back-shift operators, input and output terminals. Each input or output terminal has been paired with a back-shift operator. The maximum time-shift of the back-shift operators is three time samples, which means that the maximum possible process lag is equal to four samples. For example, *parent 2* contains a u_{k-4} term resulting from the combination of a q^{-3} operator and a u_{k-1} terminal.

Parent 1: $\hat{y}_k = u_{k-3} + \hat{y}_{k-2}u_{k-3}$ ***Parent 2:*** $\hat{y}_k = u_{k-4} - \hat{y}_{k-2}$

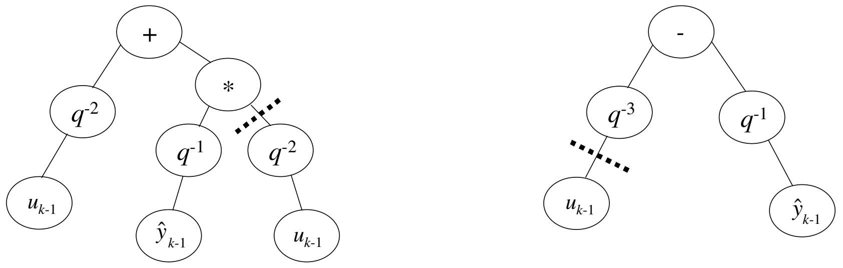


Figure 5-1 – Parent model trees

The dashed lines represent two randomly chosen crossover points. In this example, the crossover operation exchanges the '$q^{-2}u_{k-1}$' subtree from *parent 1* with the 'u_{k-1}' term from *parent 2*. The resulting offspring are shown in Figure 5-2. *Offspring 2* contains a 'u_{k-6}' term due to the application of two back-shift operators (q^{-3} and q^{-2}) on a u_{k-1} terminal.

Offspring 1**:* $\hat{y}_k = u_{k-3} + \hat{y}_{k-2}u_{k-1}$ ***Offspring 2: $\hat{y}_k = u_{k-6} - \hat{y}_{k-2}$

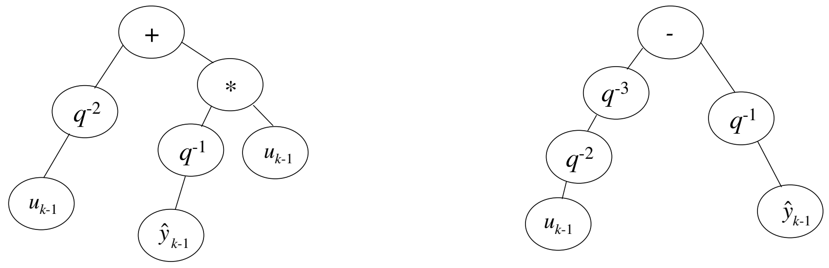


Figure 5-2 – Offspring model trees

This nesting of back-shift operators enables the algorithm to build the necessary time-shifted input and output terms without having to precisely define the number of lags at the start of the run. *Offspring 1* now contains a u_{k-1} terminal without a back-shift operator. This emphasises the reason for using u_{k-1} and $\hat{y}_{k-1}$ terminals instead of u_k and $\hat{y}_k$ – the terminals must be valid model terms if they appear without a back-shift operator. The GP algorithm would represent the model equation for *offspring 2* as follows,

Model equation: $\hat{y}_k = u_{k-6} - \hat{y}_{k-2}$

GP representation: `(q3(q2(u)))-(q1(y))`

Where, `q1, q2` and `q3` represent the back-shift operators q^{-1}, q^{-2} and q^{-3} and `u` and `y` are the input and output terminals, u_{k-1} and $\hat{y}_{k-1}$ respectively. As it is convenient to simply assign a back-shift operator to every input/output terminal appearing in a newly generated model equation, it is necessary to include an operator that does not perform a time-shift. This ensures that there is a uniform distribution of time-shifts in the initial population and allows model terms with a single process lag to be produced (e.g. `q0(u1)`$=u_{k-1}$). The GP algorithm settings and parameters are summarised in Table 5-1.

Table 5-1 – Algorithm settings and parameters

Function set	+, -, /, *, ^, `SQRT,SQR,EXP,LOG`
	Back-shift operators: `q0,q1,q2,q3`
Terminal set	Process inputs, $u_{1,k-1},...,u_{n,k-1}$ scaled in range [0 1]
	Model output, $\hat{y}_{k-1}$, $\Re$ uniformly in range [-10 10]
Crossover probability	0.7
Mutation probability	0.2
Direct reproduction probability	0.1
Generation gap	90%
Fitness measure	RMS error
Selection method	Linear ranking
Maximum tree size	500 characters

The function set contains primitives such as logarithm and exponential, meaning that the models will not be restricted to a particular type of NARX model such as the polynomial form. Apart from the terminal and function sets, the other algorithm features and settings are the same as those used for steady-state modelling. This includes the Levenberg-Marquardt non-linear least squares routine used to obtain the best possible fit for each evolved model.

5.3.2 Multiple Basis Function GP Algorithm

The MBF-GP model structure is very similar to that used for steady-state modelling. As before, each population member is a linear sum of a number of non-linear basis functions,

$$\hat{y}_k = a_0 + \sum_{j=1}^{m} a_j g_j(u_{1,k-1},...,u_{n,k-1}, \hat{y}_{k-1}, q^{-1}) \qquad 5\text{-}16$$

Where m is the number of basis functions (m was chosen as a uniformly random integer in the range [1 10]), a_j are constants and a_0 is a bias or offset term. The model structure given by equation 5-16 is the same as that used by the steady-state MBF-GP algorithm apart from inclusion of the back-shift operator and time-shifted inputs and model output.

As equation 5-16 is linear in the parameters, the constants ($a_0,\ldots,a_m$) can be optimised using the method of recursive least squares (RLS). As the auto-regressive model terms are predicted values of the output ($\hat{y}$), models must be evaluated recursively, meaning that batch least squares methods are not suitable for parameter optimisation. RLS is the recursive form of the method of ordinary least squares and is derived in the appendix. A weakness of the standard RLS algorithm is its sensitivity to computer round-off errors due to ill conditioning of the covariance matrix update. However, the numerical stability and robustness of the algorithm can be significantly improved by using the method of U-D factorisation described by Bierman (1977). Further details of the standard RLS algorithm and the modifications required for performing the U-D covariance measurement update can be found in Kanjilal, (1995).

As with the MBF-GP algorithm used for steady-state modelling, the dynamic version of the algorithm uses both high and low-level crossover. High-level crossover enables the algorithm to exchange whole basis functions and adapt the total number of functions present in each population member. Low-level crossover provides a mechanism for subtrees to be transferred between individuals.

5.4 Comparison of Results

Three case studies were used to compare the performance of the standard and MBF-GP dynamic modelling algorithms – a test system with and without a time delay and an industrial cooking extruder. Plots of the input-output data and linear models for these systems are contained in the appendix. The results are compared using the analysis procedure used in the steady-state modelling case studies.

5.4.1 Case Study 1 – Test System

The following single-input single-output non-linear test system (Narendra and Parthasarathy, 1990) was used to compare the ability of the standard and MBF-GP algorithms to evolve accurate time series predictions,

$$y_k = \frac{y_{k-1} y_{k-2} (2.5 + y_{k-1})}{1 + y_{k-1}^2 + y_{k-2}^2} + u_{k-1} \qquad 5\text{-}17$$

The input signal, *u*, was generated as a multi-level step response in the range [0 5]. Two hundred data points were used for training and a further 200 samples were used for model validation. Both algorithms were run 20 times with a population size of 25 for 25 generations. The results are summarised in Table 5-2.

Table 5-2 - Summary of validation RMS error values for test system

	Minimum	Mean	Maximum
MBF-GP	0.0028	0.0090	0.0332
Standard GP	0.0030	0.0177	0.0329

Figure 5-3 compares the validation RMS error distributions for both algorithms. Although there is only a small difference between the best models evolved by each algorithm, the MBF-GP algorithm was able generate the more accurate models overall. A one-sided K-S test performed at the 95% confidence level confirms that this difference is significant. It can be seen that the majority of the MBF-GP errors lie below 0.015 (17 of the 20 runs), whereas the standard GP errors are more evenly distributed around this value.

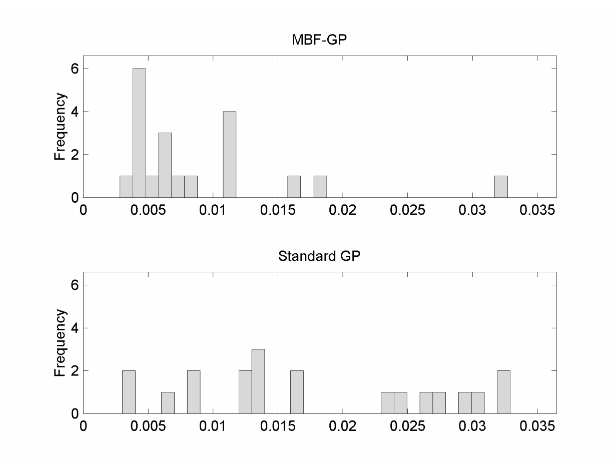


Figure 5-3 – Comparison of validation RMS error distributions (Test system).

The MBF-GP model expression with the lowest RMS error on the validation set is shown in Table 5-3. The model is presented as a set of separate basis functions to give an idea of the typical size and number of functions present in this type of model. The individual basis functions have been simplified.

Table 5-3 - Model structure with lowest RMS error on validation data set

Basis functions	Parameter values
$0.2166u_{k-1}(0.8002-u_{k-1})+u_{k-2}$	0.2267
$(2.136^{u_{k-1}})^2+\log(0.1675\hat{y}_{k-2})$	0.0216
$-8.437u_{k-1}+u_{k-1}^2-u_{k-1}\hat{y}_{k-3}+0.6177$	-0.0521
$u_{k-3}+\hat{y}_{k-2}$	0.0892
$0.5392+\hat{y}_{k-3}$	0.0277
Bias	0.0960

The model with the lowest validation RMS generated using the standard GP algorithm is shown below in simplified form,

$$\hat{y}_k=(u_{k-1}+1.161/\hat{y}_{k-3}^{0.02689\hat{y}_{k-1}})(0.4730-0.01326u_{k-2}) \qquad 5\text{-}18$$

Although the basis functions in Table 5-3 could be combined in order to allow further simplification of the model equation, the resulting model would still be more complex than the model evolved by the standard GP algorithm.

5.4.2 Case study 2 – Test System with Time Delay

This case study uses the same system equation as the first case study with the addition of a time delay of ten sample instances,

$$y_k = \frac{y_{k-1}y_{k-2}(2.5 + y_{k-1})}{1 + y_{k-1}^2 + y_{k-2}^2} + u_{k-11} \qquad 5\text{-}19$$

The system described by equation 5-19 will demonstrate the algorithms' ability to generate models that contain process lags greater than those provided by the back-shift operators in the function set. As it is likely to be more difficult for GP to develop an accurate model of this system, the population size and number of generations were increased. Each algorithm was run 20 times with a population size of 50 for 50 generations. Table 5-4 shows a summary of the results on the validation data set.

Table 5-4 – Comparison of validation RMS error values for case study 2.

	Minimum	Mean	Maximum
MBF-GP	0.0047	0.0238	0.0622
Standard GP	0.0045	0.0225	0.0505

Figure 5-4 shows the distribution of the best validation RMS values for both algorithms. It can be seen that there is no significant improvement in model accuracy to be gained by using the MBF-GP algorithm. A two-sided K-S test performed at the 95% confidence level supports this observation.

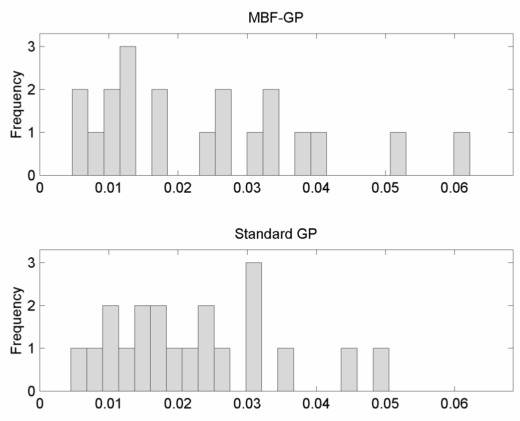


Figure 5-4 - Comparison of validation RMS values (test system with time delay)

The most accurate model on the validation data developed using the standard GP algorithm is shown below.

$$\begin{aligned}\hat{y}_k &= 0.0487\hat{y}_{k-5}(0.0357u_{k-11} + 0.336u_{k-15} - 0.0643 - 0.0357\hat{y}_{k-9} + 0.0432u_{k-13} \\ &+ \hat{y}_{k-5}) + 0.503u_{k-11} + 0.0140 + 0.0841\hat{y}_{k-3} - 0.0298\hat{y}_{k-6} + 0.432\hat{y}_{k-1}\end{aligned} \qquad 5\text{-}20$$

Equation 5-20 only makes use of the '+', '-' and '*' mathematical operators and is therefore a polynomial NARX model. It should also be noted that the model contains a greater number of process lags than the system equation; the model output and input have maximum lags of nine and fifteen samples respectively.

The MBF-GP model with the lowest validation RMS error is shown in Table 5-5. The basis functions have been included in the form that they are stored by the GP algorithm (with some simplification to improve readability). Table 5-5 illustrates how the algorithm has combined a series of back-shift operators in order to model the system time delay. For example, the first basis function contains a u_{k-11} term constructed from a combination of six back-shift operators.

Table 5-5 – MBF-GP model with lowest validation RMS error

Basis Functions	GP Representation	Parameters
$-0.4848u_{k-11} - 3.698$	`-0.4848*q1(q2(q1(q2(q1(q3(u) +q0(-9.3343e-2))))+3.8604))-1.871`	-1.068
$-0.4848u_{k-12} - 5.562$	`-0.4848*q1(q2(q1(q1(q2(q1(q3(u)+ q0(-9.334e-2))))+7.7208+ q0(-1.427e-2)))))-1.871`	-0.3091
$(\hat{y}_{k-1} + \hat{y}_{k-2})(\hat{y}_{k-2} - \exp((u_{k-4} - u_{k-3})\hat{y}_{k-2} - \hat{y}_{k-3}))$	`(q0(y)+q1(y))*(q1(y)-exp((q0(q3 (u))-q2(q0(u)))*q1(y)-q2(y)))`	-0.01907
$-0.4848\hat{y}_{k-2} + 0.4848\hat{y}_{k-1}$	`-0.4848*q1(y)+0.4848*y`	0.3091
$\hat{y}_{k-2} - 3.860$	`q1(y)+q3(-3.8604)`	0.2659
$\hat{y}_{k-2} + \hat{y}_{k-3}$	`q1(y)+q2(y)`	0.04660
0.1482	`q1(0.3849*q0(0.3849))`	-2.549
Bias	1	-4.259

Although there appears to be no advantage gained by using the MBF-GP algorithm in terms of model accuracy, the algorithm requires considerably less computational effort than the standard GP algorithm. This is illustrated by Figure 5-5, which compares model accuracy obtained with the number of FLOPs performed by each algorithm. It can be seen that, for a given RMS error, the standard GP performs a substantially greater number of calculations than the MBF-GP algorithm.

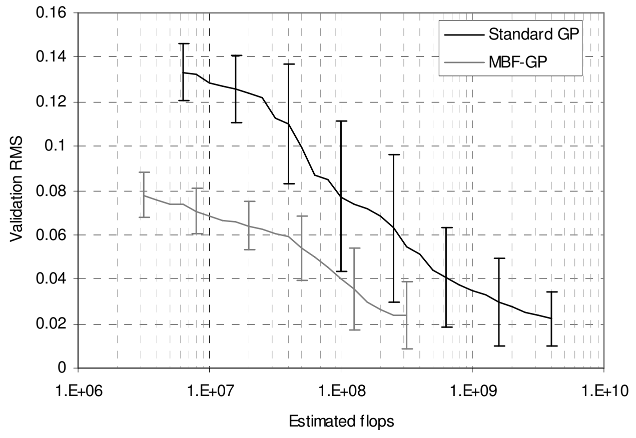


Figure 5-5 – Comparison of computational cost (test system with time delay)

This difference is a consequence of the optimisation method employed by each algorithm. The standard GP algorithm uses L-M optimisation, which requires a large number of function evaluations in order to calculate gradient information for each model parameter. On the other hand, the RLS algorithm optimises model parameters in a single pass of the data set, giving the MBF-GP algorithm an advantage in terms of computational effort.

5.4.3 Case Study 3 – Cooking Extruder

The industrial cooking extruder was described in detail in chapter 3. A data set containing 595 records of the degree of starch gelatinisation (g) together with the corresponding inputs was generated for dynamic model development. The following inputs were used - feed flowrate (Q_f), feed moisture content (M_f), screw speed (ω), and the feed temperature (T_f). A set of 400 data points was used for model training and the remaining 195 data points were used for model validation. Each algorithm was run 20 times with a population of 50 for 50 generations. The resulting validation RMS error distributions are shown in Figure 5-6.

Table 5-6 – Comparison of validation RMS error values for extruder

	Minimum	Mean	Maximum
MBF-GP	0.0348	0.0401	0.0496
Standard GP	0.0412	0.0607	0.0780

The results show that the models developed by the MBF-GP algorithm are more accurate than those obtained using the standard algorithm. All of the MBF-GP runs produced models with a validation RMS error of less than 0.05, whereas only five standard GP runs were able to achieve a similar level of performance. This conclusion is supported by one-sided K-S test (95% confidence level)

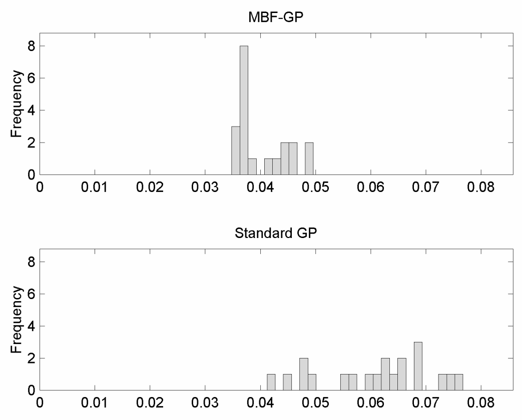


Figure 5-6 – Comparison of validation RMS distributions (extruder data).

The MBF-GP model that produced the most accurate prediction on the validation data is shown in Table 5-7. The model is constructed from nine basis functions, although each of the individual functions is relatively parsimonious.

Table 5-7 – MBF-GP model structure with lowest validation RMS

Basis functions	Parameter values
$M_{f\,k-2} + \log(0.1557^{\hat{g}_{k-3}})$	-0.2744
ω_{k-2}^2	0.1448
$(\omega_{k-1} + \hat{g}_{k-3}^{1/2})/(T_{f\,k-2} + M_{f\,k-2}) - \omega_{k-2}$	0.0023
$Q_{f\,k-1} + \omega_{k-2}$	0.1829
$0.08779\omega_{k-2}$	3.0189
$(Q_{f\,k-3} + Q_{f\,k-3}^2)^{1/2}$	-0.2387
$-0.5462M_{f\,k-2}$	-0.4214
$\hat{g}_{k-3}\omega_{k-2}$	-0.5323
$\hat{g}_{k-2}^2$	0.2766
Bias	0.2182

The model with the lowest validation RMS generated by the standard GP algorithm is shown in simplified form below,

$$\hat{g}_k = \log(0.4425Q_{f_{k-3}} + 0.06713M_{f_{k-1}} - 1.248\hat{g}_{k-3} - 1.216) - 0.4519Q_{f_{k-3}}^{1/2} + 0.2681\omega_{k-2} - (1.125e-4)/Q_{f_{k-3}}^{2} \quad \text{5-21}$$

The predictions for the degree of starch gelatinisation generated using these models are shown in Figure 5-7 and Figure 5-8. The plots show that both models accurately represent the dynamics of the process, however, as would be expected with a more complex application, are subject to a higher degree of inaccuracy than the test systems studied earlier. It can also be seen that the MBF-GP model gives a more accurate prediction on the validation data, having an RMS error of 0.0301 compared to 0.0412 for the standard GP model.

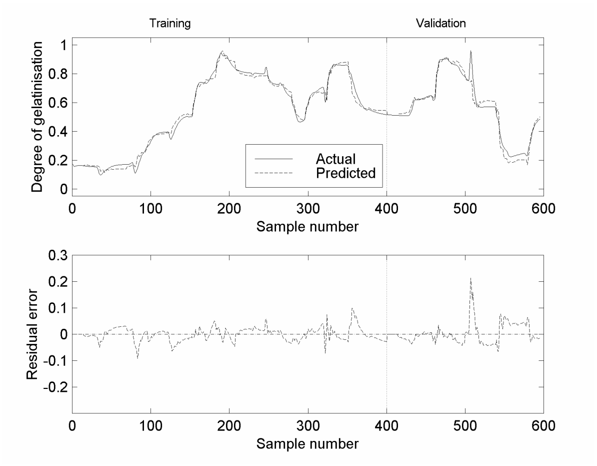


Figure 5-7 - Prediction for degree of starch gelatinisation (Standard GP)

The models described by equation 5-21 and Table 5-7 make little or no reference to the extruder feed temperature (T_f). This was also a feature of the steady-state extruder models developed in chapter 3. This can be explained by the fact that the fluctuations in feed temperature cover a relatively small range (27.5-32.5 C) and are insignificant when compared to the temperature increase that takes place inside the extruder.

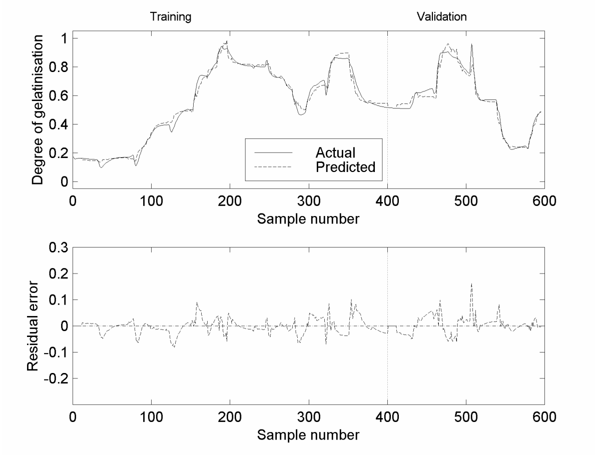


Figure 5-8 –Prediction for degree of starch gelatinisation (MBF-GP)

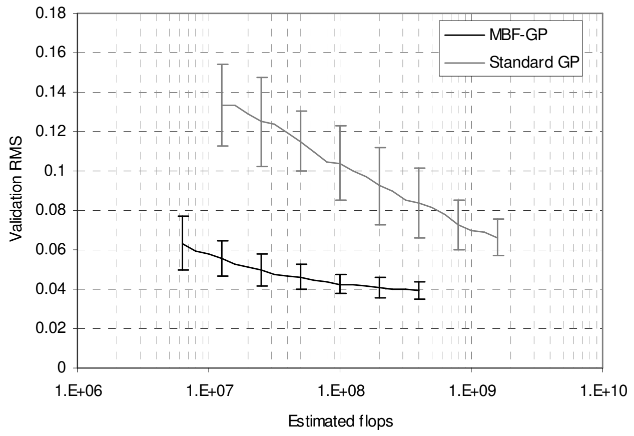


Figure 5-9 – Comparison of computational cost (extruder data)

Figure 5-9 shows that, in terms of computational cost, the MBF-GP algorithm is much more economical than the standard algorithm. The narrower error bars support the observation that the MBF-GP algorithm produces more consistent results than the standard GP algorithm.

5.5 Comparison with Neural Networks

In the previous chapter it was demonstrated how neural networks could be used for steady-state process modelling and the results were compared to those obtained using the MBF-GP algorithm. The same network structure can be modified in a number of ways in order to allow the development of dynamic process models.

5.5.1 Dynamic Modelling Using Neural Networks

The simplest approach is to use a time history of input variables ($u_{k-1}, u_{k-2}, ..., u_{k-\tau}$) as inputs to the network, so that each network input consists of a process input shifted back in time (Bhat and McAvoy, 1989). This is similar to the FIR approach (equation 5-6) and has the disadvantage that a long time history of inputs may be required to enable the network to accurately capture the dynamics of the process. This means that the network will have a large number of inputs, giving rise to a complex model with a large number of parameters to be optimised. This will increase network training times and mean that there is a higher probability of the network converging around local minima.

A possible solution is to configure the network in the form of a NARX model by assigning inputs that are past values of the process input(s) and output ($u_{k-1}, u_{k-2}, ..., u_{k-\tau_u}, y_{k-1}, y_{k-2}, ..., y_{k-\tau_y}$). This approach has been applied to system identification problems using conventional feedforward (Chen *et al.*, 1990b) and radial basis function networks (Chen *et al.*, 1990a). In order to enable the network to predict the output for more than a single time step in the future, the process output must be replaced by the network output, $\hat{y}$. The resulting network structure is known as a globally recurrent network. One of the drawbacks of this approach is that training times can be high (Turner *et al.*, 1996).

An alternative approach is the filter based neural network (FBNN) (Willis *et al.*, 1992), where the hidden layer neurons of the conventional feedforward network are augmented with simple linear dynamic processing capabilities. Although it may be possible to include neurons with any dynamic characteristic, Willis *et al.* (1992) suggest that simple first order transfer functions or ‘filters’ should be sufficient for most applications. The structure of a typical filter-based neural network is shown in Figure 5-10 (in practice, the filters are implemented in discrete time using difference equations).

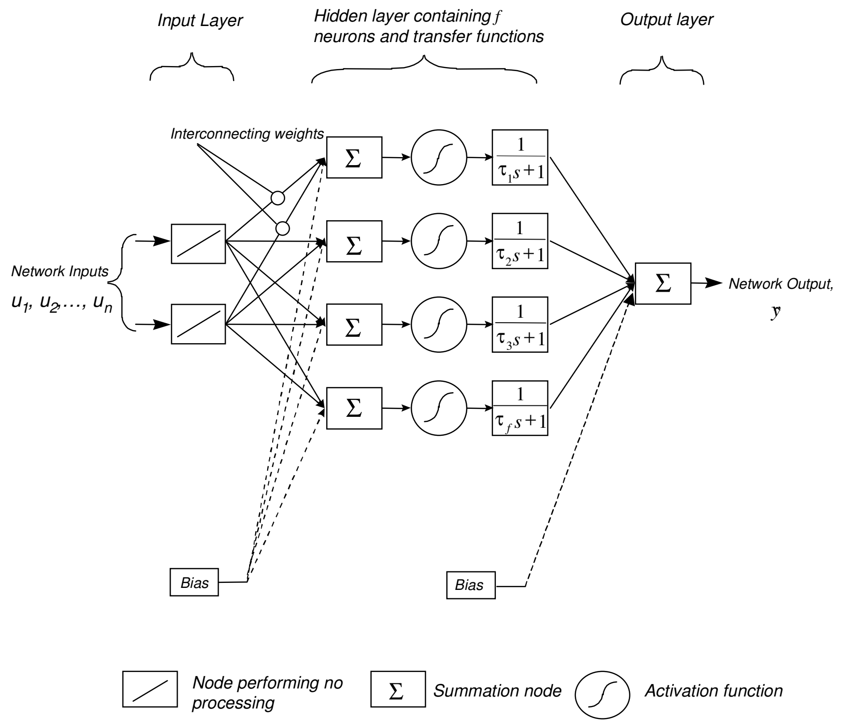


Figure 5-10 – Filter based neural network architecture

The process of training filter-based neural networks requires the optimisation of the filter constants $(\tau_1, \tau_2, ..., \tau_f)$ as well as the weights and bias values. This means that the back-propagation training algorithm used in chapter 4 cannot be used for FBNN training without modification. Consequently, a L-M algorithm was used for network training. Lennox (1996) used this technique to develop models of several chemical processes and found that the FBNN outperformed globally recurrent and RBF network structures in terms of the predication accuracy of the resulting models. Another advantage was that the FBNN did not need to be presented with a time history of inputs or the network output.

In order to find the network configuration giving the highest performance for each of the test cases, multiple runs were performed using different neural network architectures. The search was limited to networks consisting of a single hidden layer of 3, 5, 7, 9, 11, 13 and 15 neurons, with each network being trained 20 times. Further runs were performed if there was evidence that the minimum prediction error could be provided using a network with a different number of hidden layer neurons. To enable a fair comparison with GP to be made, the 'best' model was found by using the sum of the training and validation RMS values.

5.5.2 Test System

The results of the runs performed with various FBNN architectures on the test system data are summarised in Table 5-8.

Table 5-8 – Summary of neural network validation RMS errors

Hidden layer neurons	Minimum	Mean	Maximum
3	0.003212	0.008470	0.020129
5	0.002408	0.003389	0.019932
7	0.002405	0.002509	0.002703
9	0.002424	0.002522	0.002665
11	0.002439	0.002536	0.002712
13	0.002418	0.002533	0.002736
15	0.002426	0.002535	0.002742

Table 5-8 shows how the accuracy of the predictions improves as the network size is increased from three to seven hidden layer neurons. However, larger networks do not produce any further improvement in model accuracy. Consequently, the network with seven hidden layer neurons was selected for comparison with GP. As the GP algorithm runs performed with a population of 25 individuals for 25 generations (see section 5.4.1) failed to generate models that matched the accuracy of the neural networks, the population size was increased to 50 and the duration of the runs was extended to 50 generations. Table 5-9 and Figure 5-11 compare the RMS values obtained using the neural network and MBF-GP algorithms.

Table 5-9 – Comparison of validation RMS values (test system)

	Minimum	Mean	Maximum
MBF-GP	0.0018	0.0028	0.0046
FBNN (1-7-1)	0.0024	0.0025	0.0027

The results show that the RMS errors produced by the neural network form a narrower distribution than those obtained using the MBF-GP algorithm. This means that, although the most accurate model was produced by the MBF-GP algorithm, the neural network appears to be the more consistent of the two approaches. A two-sided K-S test performed at the 95% confidence level confirms that the difference between the distributions is significant.

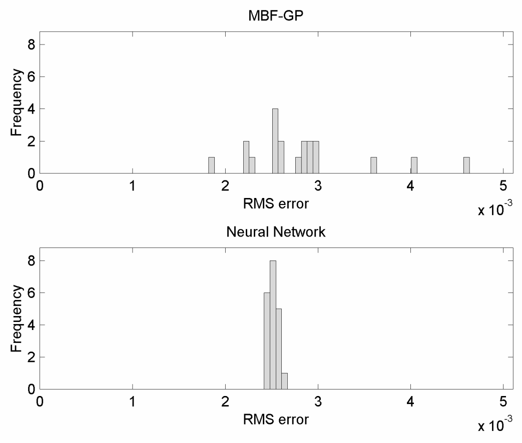


Figure 5-11 – Comparison of validation RMS values (test system)

Although several of the GP algorithm runs produced RMS errors that were inferior to the FBNN results, the accuracy of the predictions is still very high.

5.5.3 Test System with Time Delay

In this case study, it is difficult to make an unbiased comparison between the neural network and MBF-GP algorithms. The FBNN does not have the ability to model the system time delay and will be at a disadvantage when compared to the GP algorithm. The easiest way to overcome this problem is to present the network with input data that has the time delay removed. However, this will hand the advantage to the neural network, as the GP algorithm has to use a combination of back-shift operators and u_{k-1} terminals to identify the time delay. One solution is to carry out two batches of network runs, with and without the time delay removed, and observe the relative performance of the GP algorithm. The results of these runs are summarised in Table 5-10 and Table 5-11

Table 5-10 – Summary of FBNN results (no time delay compensation)

Hidden layer neurons	Minimum	Mean	Maximum
3	0.0451	0.0509	0.0589
5	0.0442	0.0525	0.0551
7	0.0387	0.0494	0.0549
9	0.0390	0.0477	0.0549
11	0.0410	0.0486	0.0538
13	0.0473	0.0488	0.0538
15	0.0417	0.0480	0.0524

Table 5-11 –Summary of FBNN results (time delay removed)

Hidden layer neurons	Minimum	Mean	Maximum
3	0.00913	0.01079	0.01601
5	0.00399	0.00993	0.01403
7	0.00313	0.00746	0.01404
9	0.00232	0.00732	0.01443
11	0.00206	0.00702	0.01404
13	0.00332	0.00642	0.01057
15	0.00281	0.00813	0.01540

In each case, the 'best' network architecture was selected using the mean validation RMS values. Although other architectures may provide 'one-off' results that outperform these networks, it was thought that networks with the lowest mean RMS provided the best compromise. The neural network and GP algorithm results are compared in Table 5-12 and Figure 5-12.

Table 5-12 – Comparison of validation RMS values

	Minimum	Mean	Maximum
MBF-GP	0.0047	0.0238	0.0622
FBNN#1 (1-9-1)	0.0390	0.0477	0.0549
FBNN#2 (1-13-1)	0.0033	0.0064	0.0106

As expected, the neural network with time delay compensation (FBNN#2) easily outperforms the network that does not have the time delay removed (FBNN#1). This observation is supported by a one-sided K-S test conducted at the 95% confidence level. The GP algorithm produced a wide range of prediction errors, with the worst values lying in the same region as those generated by FBNN#1 and the lowest errors approaching the accuracy of FBNN#2.

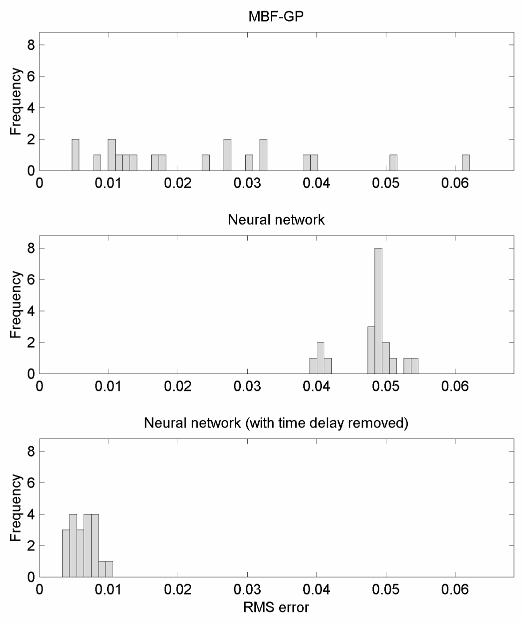


Figure 5-12 – Validation RMS error distributions (test system with time delay)

Although the GP algorithm was able to achieve performance comparable to FBNN#2 in a fraction of the runs, the algorithm also produced unacceptably poor results. This should be expected, as the algorithm has the difficult task of evolving a suitable model structure from elementary building blocks. The fact that the process time delay does not have to be explicitly accounted for gives the GP algorithm a distinct advantage over neural networks in this case study.

5.5.4 Cooking extruder

The validation RMS errors obtained using a range of different FBNN architectures on the extruder data are summarised in Table 5-13. The network containing four hidden layer neurons was selected for comparison with the MBF-GP algorithm as it produced the lowest mean RMS error.

Table 5-13 – Summary of neural network validation RMS errors

Hidden layer neurons	Minimum	Mean	Maximum
3	0.0349	0.0409	0.0565
4	0.0345	0.0382	0.0455
5	0.0338	0.0393	0.0560
7	0.0344	0.0415	0.0560
9	0.0361	0.0440	0.0571
10	0.0353	0.0406	0.0563
11	0.0367	0.0445	0.0584
13	0.0338	0.0415	0.0530
15	0.0328	0.0445	0.0692

The validation RMS values obtained using the neural network and MBF-GP algorithms are compared in Table 5-14.

Table 5-14 – Comparison of validation RMS error values for extruder.

	Minimum	Mean	Maximum
MBF-GP	0.0348	0.0401	0.0496
Neural Network (1-4-1)	0.0345	0.0382	0.0455

Figure 5-13 shows validation RMS error distributions obtained using the neural network and MBF-GP algorithms. Although the minimum, mean and maximum RMS values are slightly lower for the neural network, a one-sided (at the 95% confidence level) indicates that the difference is not significant.

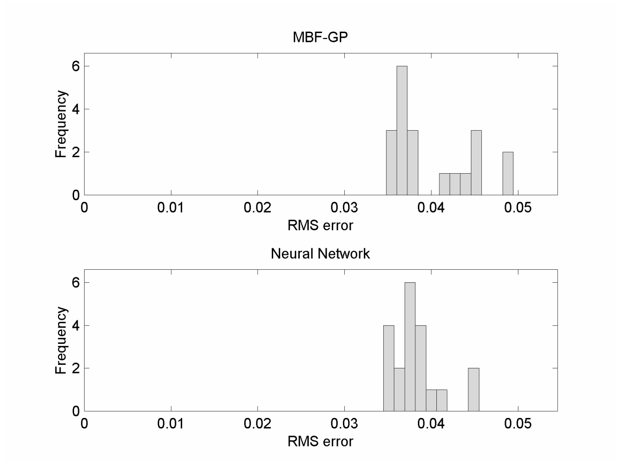


Figure 5-13 – Validation RMS error distributions (extruder data)

Figure 5-14 compares the performance of the neural network and MBF-GP algorithms in terms of the computational effort required to achieve a given validation RMS error. At higher RMS error values, the MBF-GP algorithm outperforms the neural network. This could be because the RLS optimisation routine enables the MBF-GP algorithm to raise the performance of the initial population members to a reasonable level without consuming a particularly large amount of processing power. The neural network may initially produce poor predictions due the large number of model parameters, which all have to be initialised probabilistically.

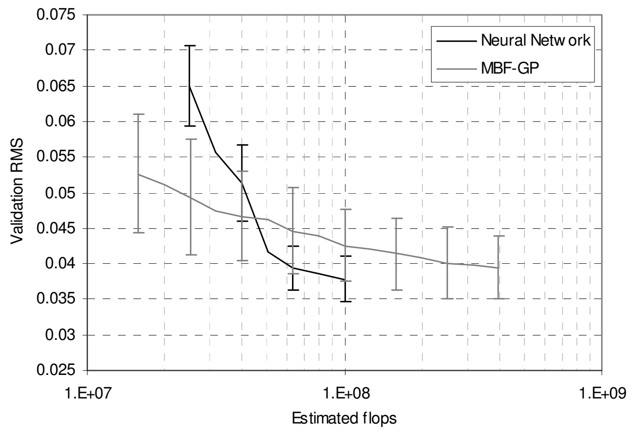


Figure 5-14 – Comparison of computational effort required by FBNN and MBF-GP algorithms (extruder data)

As the validation RMS error decreases, the neural network begins to outperform the MBF-GP algorithm, requiring fewer FLOPs to achieve the same RMS error. The difference between the algorithms continues to increase as the validation RMS errors are reduced. This could be because neural network training is essentially a parameter optimisation exercise. The GP algorithm has to explore a wide range of different model structures, performing parameter optimisation on each candidate solution, and is therefore unlikely to be as efficient as the neural network. One of the disadvantages of using neural networks is that a wide range of network architectures has to be investigated in order to obtain the best set of model predictions. If the computational effort required to carry out these additional runs is taken into consideration, the difference between the algorithms is less significant and GP becomes a more attractive possibility.

5.6 Conclusions

In this chapter, it was shown how the standard and MBF-GP algorithms could be used to evolve discrete-time models of dynamic systems. As was observed in the steady-state modelling comparison, the MBF-GP algorithm was generally able to generate models that gave more accurate predictions than the standard algorithm and required less computational effort. An interesting feature of this technique is that the algorithm can make use of past values of model terms that are not specified explicitly by the function and terminal sets. This was demonstrated in the second case study, where the algorithm combined a number of back-shift operators in order to model the system time delay. The model structures are relatively complex but do not appear to be as unwieldy as those evolved by the steady-state modelling algorithms. This may be because back-shift operators occupy a considerable proportion of each model string. These sections are easily simplified, leading to a greater reduction in model size.

The results also revealed that the performance of GP compared to neural networks was system dependent. The neural network produced the more consistent results on the first test system, although the GP algorithm evolved the most accurate prediction. On the second test system, the neural network was only able to outperform GP once the system time delay has been manually removed from the input data. There was little difference between the two techniques in terms of model accuracy on the extruder case study. However, the GP algorithm required more computational effort to achieve the same accuracy as the neural network.

For real problems, additional factors must be taken into account when assessing the suitability of a model derived using GP. Different applications may raise issues that make a neural network the more desirable option. For example, a reason offered by some researchers (Chen *et al.*, 1990b) as justification for choosing a neural network model is that the network output is bounded. This is beneficial, as the network will behave less erratically when applied to data lying outside of the range used for training. The complexity of GP models and the probabilistic nature of model evolution mean that a GP model cannot be expected to behave as predictably. However, Hernandez and Yarkun (1993) describe a possible solution to this problem, which involves the use of a NARMAX model combined with a sigmoid function. The GP algorithm could be modified in a similar way, if this model property was thought to be desirable.

Another important aspect of neural network and GP models is their robustness to process noise. Noise was not added to any of the data sets used in this work, as the aim was to concentrate on the differences in prediction accuracy achieved by each modelling technique. However, in practice, data sets typically contain noisy process measurements and it would be useful to carry out a comparison between the algorithms when subjected to these conditions. Possible issues concerning the MBF-GP algorithm include the robustness of the RLS routine, which may produce biased parameter estimates under certain conditions.

Finally, the use of neural networks for non-linear modelling is well established and supported by research into a wide range of theoretical and practical applications. This cannot be said for GP, although the number of applications to engineering problems has increased in recent years. Also, unlike neural networks, which have a generic model structure, GP models vary greatly from one algorithm run to the next and are more of an unknown quantity. These factors, combined with the fact that the tools required to develop neural network models are more widely available, mean that most engineers will turn to neural networks before considering GP for model development. However, one area in which evolutionary algorithms such as GP are being used extensively is that of multi-objective problem solving. The steps that must be taken to apply the GP algorithm to this type of problem are described in the next chapter

6 Multi-objective Genetic Programming

6.1 Introduction

In the previous chapter, it was shown how a MBF-GP algorithm could be used to evolve models of dynamic chemical processes. Comparison with a 'standard' GP algorithm revealed that the MBF-GP algorithm produced a higher level of performance in terms of model accuracy and computational effort required. All of the GP runs in previous chapters were concerned solely with the minimisation of the RMS error between the actual and predicted process output. However, process model development is a task that may require a number of other factors or 'objectives' to be considered before the final solution is reached. Some examples of possible objectives are,

- Measures of model parsimony, for example, the number of model parameters and the maximum number of process lags.
- Additional or alternative measures of prediction error such as residual variance, one-step ahead and long term prediction errors.
- Model validation criteria such as residual correlation tests and statistical information criteria.

This chapter demonstrates how the MBF-GP algorithm can be modified to incorporate additional measures of model performance. The next chapter compares the algorithm with the single objective algorithm to assess the advantages gained by considering extra objectives during the process of model evolution.

6.2 Multi-objective Evolutionary Algorithms

Although the first notable work on multi-objective evolutionary algorithms (MOEAs) was published by Schaffer in the mid 1980s (Schaffer, 1985), it was not until a decade later that there was a substantial increase in the number of applications of such algorithms to engineering problems. Applications including controller design (Chipperfield and Fleming, 1995), system identification (Fonseca and Fleming, 1996a), process optimisation (Garg and Gupta, 1999) and scheduling (Shaw *et al.*, 1999) have all received attention in recent years. While the majority of these applications are essentially optimisation problems using multi-objective genetic algorithms (MOGAs), the fundamental concepts are also applicable to other evolutionary algorithms including GP.

Rodríguez-Vázquez and Fleming (1998) used a multi-objective genetic programming (MOGP) approach to system identification, making use of additional model performance measures such as the number of process lags and linear correlation criteria. The MOGP algorithm used the same multi-objective ranking and fitness assignment scheme employed by the MOGA used in the authors' previous work. An overview of the important features and applications of MOEAs can be found in Fonseca and Fleming (1995), Coello (1999), and Van Veldhuizen and Lamont (2000). The remainder of this section outlines the aspects of MOEAs relevant to the MOGP algorithm used in this thesis.

The practice of solving multi-objective engineering problems can prove to be a difficult and time-consuming task. Whereas single objective problems may have unique optimal solutions, multi-objective problems often have a large number of possible solutions. This is because the different performance measures that characterise the multi-objective problem may conflict with each other, meaning that only a partial ordering of the search space is possible. The solution will therefore be in the form of a set of individuals representing a trade-off between different levels of performance in each objective domain.

The final solution generated by a MOEA can be seen as the result of both an *evolutionary* and a *decision* process. The evolutionary aspect of the algorithm enables the search to cover a diverse range of possible solutions with the aim of achieving improved performance relative to the objectives under consideration. However, the nature of the multi-objective problem means that the algorithm is likely to produce a set of candidate solutions. The final solution must then be chosen from this set of individuals. This decision process can take place at different stages in the algorithm run. Hwang and Masud (1979) suggested the following categories,

***A priori* preference articulation**. The multi-objective problem is effectively transformed into a single objective problem. The weighted sum approach (section 6.2.1.2) is an example of this technique.

Progressive preference articulation. Decision-making takes place as the evolutionary process proceeds. Each generation presents a new set of candidate individuals to be considered. For example, Fonseca and Fleming (1998) proposed a method for progressive articulation of preferences within a MOGA framework (see section 6.2.2).

***A posteriori* preference articulation**. At the end of the search, the algorithm presents a set of candidate solutions from which the desired solution is chosen.

A disadvantage of *a priori* preference articulation is that an inappropriate choice of parameters (e.g. cost function weights) can lead to the discovery of an unsuitable solution. Additional algorithm runs will then have to be performed using different parameters values. This is in contrast to the *a posteriori* method, which will ideally lead to a Pareto optimal (see section 6.2.2) set of solutions from which to choose. This can be advantageous as no possible solutions are discarded until the final decision stage. However, for real life problems, the trade-off surface between the objectives can be extremely complex and it may be beneficial for the search to be directed towards the region that it of most interest. In this case, progressively articulating preferences may yield better results.

A wide range of methods has been used to apply evolutionary algorithms to the task of multi-objective problem solving. The most significant are outlined below, along with their comparative advantages and disadvantages. The techniques are grouped into two categories – Pareto and non-Pareto approaches.

6.2.1 Non-Pareto approaches

6.2.1.1 The Vector Evaluated Genetic Algorithm

It is widely recognized that Schaffer (1985), was the first to demonstrate the potential of evolutionary algorithms to discover a set of non-dominated solutions in a single algorithm run. The resulting algorithm, known as the vector evaluated genetic algorithm (VEGA), considered each of the objectives separately by dividing the population into separate parts, each corresponding to a different objective. Individuals were then chosen from each section of the population based on their performance with respect to only one objective. These where then shuffled and the genetic operators used to create the next population in the usual manner. One of the drawbacks of this method was that the algorithm tended to produce individuals that performed extremely well in a single objective and poorly in all of the others. This phenomenon is known as speciation and is a direct result of the selection process being based on an individual's performance with respect to a single objective.

6.2.1.2 Aggregating approaches

Another technique used to handle multiple objectives is to combine the individual objectives using a weighted cost function of the form,

$$F_i = \sum_{j=1}^{n} w_j f_j(x_i) \qquad 6\text{-}1$$

Where F_i is the fitness of population member x_i, w_j are the weighting coefficients representing the relative importance of each of the n objectives and f_j are the functions used to generate the objective values. This approach can be problematic for a number of reasons. For instance, insufficient knowledge of the problem may make it difficult to assess the relative importance of each objective. This will make it difficult to select appropriate values for the weighting coefficients in the cost function. If algorithm performance is deemed unsatisfactory, the weightings must be adjusted and more runs performed until a satisfactory solution is achieved. This may prove to be time consuming, especially if algorithm performance is very sensitive to small changes in these parameters. In addition, objective values must be appropriately scaled before they can be combined to form the cost function. The method of scaling may alter the shape of the trade-off surface and make it more difficult or even impossible to find a suitable balance (Fonseca and Fleming, 1995).

Examples of MOGAs making use of the weighted sum approach include Hajela and Lin (1992), and Ishibuchi and Murata (1996). Hajela and Lin's Genetic Algorithm (HLGA) addresses some of the drawbacks of the weighted sum technique by adaptively changing the cost function weightings as the run proceeds. This is achieved by encoding the weight values as part of the genotype. Although Fonseca and Fleming (1997) reported that linear fitness combination was the most popular MOEA technique, Pareto-based techniques have become increasingly popular in recent years. This is especially true for real world scientific and engineering applications to which 90% of Pareto based MOEA applications are applied (Van Veldhuizen and Lamont, 2000). Pareto based methods are discussed in the next section.

6.2.2 Pareto-based Approaches

Some of the problems associated with non-Pareto methods may be avoided by comparing individuals using the concept of Pareto dominance, defined as follows,

Assuming a minimisation problem, a vector $\mathbf{v} = [v_1, \ldots v_n]$ is said to dominate vector $\mathbf{u} = [u_1, \ldots u_n]$ if it is partially less than **u**, i.e. the following criteria must be satisfied,

$$\forall i \in \{1, \ldots, n\}, v_i \le u_i \quad \wedge \quad \exists i \in \{1, \ldots, n\}, v_i < u_i \qquad \text{6-2}$$

The family of solutions to a multi-objective optimisation problem is said to be Pareto-optimal if, for each individual, an improvement in performance in one objective dimension cannot be achieved without degrading performance with respect to other objectives. Pareto based fitness assignment was first proposed by Goldberg (1989) as a means of assigning equal probabilities of selection to all non-dominated individuals in the current population.

An important aspect of Pareto based ranking is that credit is given to an individual that has a high level of performance in one objective, even if it performs badly with respect to all other objectives. Also, unlike methods such as the weighted sum approach, Pareto based ranking is independent of the scaling of the objectives since raw objective values may be used when comparing the performance of individual population members.

6.2.2.1 Preference information

The set of Pareto-optimal solutions for a particular problem may be very large and it will therefore be difficult to effectively sample all regions of the trade-off surface using a GP algorithm that has a relatively small population size. As the number of objectives increases, the trade-off surface becomes more complex and it becomes less likely that the algorithm will be able to find solutions that perform acceptably with respect to each objective.

In addition, some objectives may be much easier to minimise (or maximise) than others and this may cause the population to become saturated with individuals that perform acceptably in only these objectives. Applying genetic operators to these individuals is likely to create even more individuals that have a high level of performance in only one objective. As the evolutionary process continues, the population may converge to a set of solutions that perform extremely well in one objective and poorly in all of the others. These problems can be counteracted by using the goal based Pareto ranking method proposed by Fonseca and Fleming (1998). This enables the user to specify desired levels of performance in each objective domain and direct the search towards the required region of the trade-off surface.

This approach requires goal values to be specified for each of the objectives being considered. These values are then used to selectively eliminate objectives that have reached a certain level of performance from the comparison procedure. For example, if two population members satisfy the same goals, only objectives that do not satisfy their goals are used to determine which individual dominates. This concentrates the search in the direction of the objectives that have not achieved the desired level of performance and prevents the algorithm from attempting to further minimise objectives that have already attained acceptable values. Alternatively, if objectives are allocated goals that are unattainable, those objectives will always be included in the comparison process. In the special case that all goals are unattainable, the ranking procedure becomes equivalent to conventional Pareto ranking. The method described by Fonseca and Fleming ranks populations of individuals by comparing them using the *preferability* relationship defined below.

Consider two objective vectors $\mathbf{u} = [u_1, u_2, ..., u_n]$ and $\mathbf{v} = [v_1, v_2, ..., v_n]$, and a goal vector $\mathbf{g}$ containing goal values for each objective, $\mathbf{g} = [g_1, g_2, ..., g_n]$.

If $\mathbf{u}^{\overset{u}{\smile}}$ refers to the components of $\mathbf{u}$ that satisfy their goals and $\mathbf{u}^{\overset{u}{\frown}}$ refers to the components of $\mathbf{u}$ that violate their goals, a vector $\mathbf{u}$ is said to be *preferable* to another vector $\mathbf{v}$ given a goal vector $\mathbf{g}$ if,

$$(\mathbf{u}^{\overset{u}{\frown}} \prec \mathbf{v}^{\overset{u}{\frown}}) \quad \vee \quad \left\{ (\mathbf{u}^{\overset{u}{\frown}} = \mathbf{v}^{\overset{u}{\frown}}) \wedge \left[(\mathbf{v}^{\overset{u}{\smile}} \not\leq \mathbf{g}^{\overset{u}{\smile}}) \vee (\mathbf{u}^{\overset{u}{\smile}} \prec \mathbf{v}^{\overset{u}{\smile}}) \right] \right\} \qquad \text{6-3}$$

Where $\mathbf{u} \prec \mathbf{v}$ denotes that $\mathbf{u}$ dominates $\mathbf{v}$. $\mathbf{v}^{\overset{u}{\frown}}$ refers to the components in $\mathbf{v}$ that correspond to the objectives in $\mathbf{u}$ that violate their goals. For example if u_1 and u_2 violate their goals, $\mathbf{v}^{\overset{u}{\frown}}$ refers to v_1 and v_2.

It is important to note that the inequalities in equation 6-3 apply to the individual components of the vectors. For example, we can refer to the components of $\mathbf{u}$ that satisfy their goals by using the following inequality,

$$\mathbf{u}^{\overset{u}{\smile}} \leq \mathbf{g}^{\overset{u}{\smile}} \qquad \text{6-4}$$

This can be re-written in terms of the individual vector components, i.e.,

$$\forall i \in \overset{\mathrm{u}}{\mathbf{u}}{}^{\smile},\;\; u_i \le g_i \qquad \text{6-5}$$

Equation 6-3 describes three possible conditions, one of which must be satisfied for **u** to dominate **v,**

- The components of **u** that do not satisfy their goals dominate the corresponding components of **v**: $(\overset{\mathrm{u}}{\mathbf{u}}{}^{\frown} \prec \overset{\mathrm{u}}{\mathbf{v}}{}^{\frown})$.
- The components of **u** that do not satisfy their goals are equal to the corresponding components of **v**, but **v** has at least one other component that does not satisfy its goal: $(\overset{\mathrm{u}}{\mathbf{u}}{}^{\frown} = \overset{\mathrm{u}}{\mathbf{v}}{}^{\frown}) \wedge (\overset{\mathrm{u}}{\mathbf{v}}{}^{\smile} \nleq \overset{\mathrm{u}}{\mathbf{g}}{}^{\smile})$.
- The components of **u** that do not satisfy their goals are equal to the corresponding components of **v**, but **u** dominates **v** as a whole: $(\overset{\mathrm{u}}{\mathbf{u}}{}^{\frown} = \overset{\mathrm{u}}{\mathbf{v}}{}^{\frown}) \wedge (\overset{\mathrm{u}}{\mathbf{u}}{}^{\smile} \prec \overset{\mathrm{u}}{\mathbf{v}}{}^{\smile})$.

Note that equation 6-3 assumes that the aim is to *minimise* all of the objectives. Therefore, the components of a vector must be less than the corresponding goal values in order to satisfy them.

6.2.2.2 Fitness assignment

After all individuals have been compared with each other using the preferability relationship described by equation 6-3, Fonseca and Fleming (1998) propose the following ranking and fitness assignment scheme. Ranking is assigned according to the number of individuals that dominate each population member. Consequently, all non-dominated individuals are given an equal ranking. If the set of non-dominated individuals is assigned rank 0, the ranking of population member x_i dominated by p_i^t individuals at generation t is given by,

$$rank(x_i, t) = p_i^t \qquad \text{6-6}$$

The individuals are then sorted by rank and fitness allocated by interpolating linearly between the best and the worst individuals. Fitness values must be averaged for equally ranked individuals to ensure that they are given equal chance of reproduction and maintain a constant level of selection pressure across the population.

Figure 6-1 and Figure 6-2 demonstrate how (for a bi-objective problem) the ranking assigned to population members is affected by the inclusion of goal values. Figure 6-1 shows how a small population of individuals would be ranked using Pareto ranking (non-dominated individuals are circled). The non-dominated individuals are assigned a rank of zero and all other individuals are ranked according to the number of population members that dominate them.

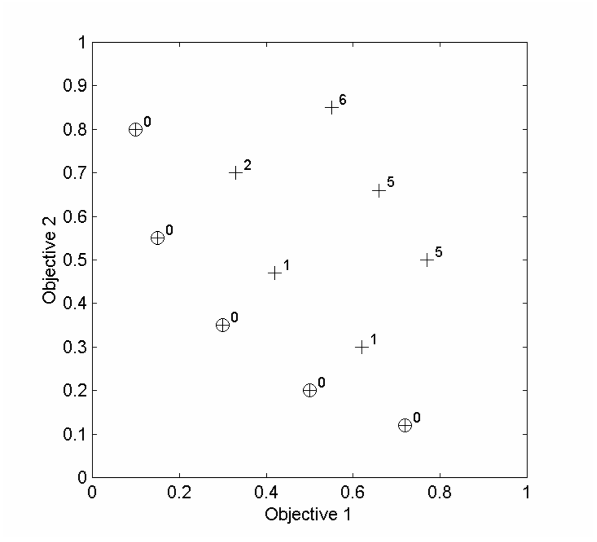


Figure 6-1 – Pareto ranking

Figure 6-2 shows the ranks assigned to the same individuals by performing Pareto based ranking with goals using the preferability relation described earlier. As only one individual satisfies both goals, this individual is said to be preferable to the others and is assigned a rank of zero. The figure shows how the solutions in the area marked 'A' are ranked in terms of objective 2 only, since they have all attained the desired level of performance with respect to objective 1 (g_1).

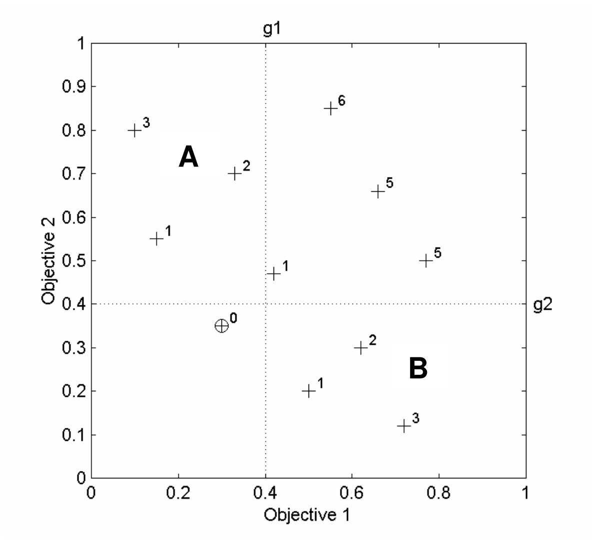


Figure 6-2 – Pareto ranking with goals.

In the region marked 'B', all individuals have satisfied the goal g_2 and are ranked using their objective 1 values only. This is beneficial, as search effort is not wasted by attempting to minimise objectives that have already reached the required level of performance.

6.2.2.3 Priority levels

The goal-based ranking strategy also provides a means of assigning different priority levels to each of the objectives. This enables the user to incorporate preferences concerning the relative importance of the objectives into the search process. The goal values for objectives with the highest priority level must be satisfied before the objectives and goal values at the next priority level are considered. The process is repeated until the lowest priority goals are satisfied.

Figure 6-3 demonstrates how the ranking given to population members is modified by the addition of priority levels. Here, the goal value for objective 1 has been given a higher priority than objective 2 goal. Individuals that do not satisfy g_1 are ranked in terms of objective 1 only. Once individuals have achieved the desired level of performance with respect to objective 1, ranking is carried using objective 2 values only.

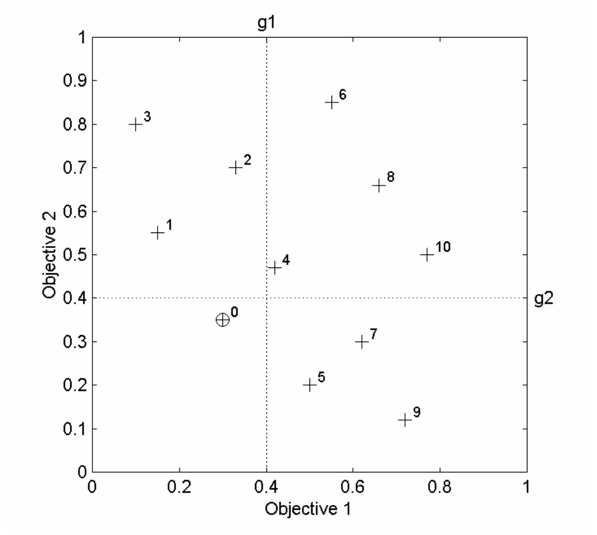


Figure 6-3 – Population ranking with goals and priorities

(g_1 has priority over g_2)

The techniques outlined above can be used in conjunction with a MOEA that uses progressive preference articulation. This allows the user to interact with the algorithm at each generation to modify goals and/or priorities in an attempt to guide the search towards a desirable solution. For example, Rodríguez-Vázquez and Fleming (1998) used this technique to evolve polynomial NARMAX models of dynamic systems using a GP algorithm. The main disadvantage of this method is that the operator must be present for the duration of the algorithm run. This is particularly relevant to the experimental procedure used in this thesis, which involves the use of multiple runs in order to make a fair comparison between algorithms. In addition, frequent user interaction is undesirable if the aim is to use GP to *automatically* generate model structures.

As a result, progressive preference articulation was not used in this thesis and each set of algorithm runs was carried out using goal/priority information fixed for the entire run length. Although goal and priority information must be specified before the algorithm run commences, this method should not necessarily suffer the same drawbacks as other methods that use *a priori* parameter specification (such as the weighted cost function approach). The algorithm will still produce a non-dominated or preferable set of solutions from which to choose. Ideally, these solutions will be more densely concentrated around the desired region of the objective space than solutions that are evolved using conventional Pareto ranking.

6.2.3 Multi-Objective GP Algorithm Details

The MOGP algorithm used in this thesis is based on the MBF-GP algorithm described in chapter 5. It was shown how the algorithm could be used to evolve accurate time series predictions of dynamic processes, requiring less computational effort than a 'standard' GP algorithm. The ranking and fitness assignment techniques are described in sections 6.2.2.1-6.2.2.3. The remaining differences between the MOGP and SOGP algorithms are outlined below.

6.2.3.1 Fitness sharing

Although evolutionary algorithms such as GP are capable of simultaneously exploring different regions of the solution space, genetic drift will cause the algorithm to eventually converge around one region of the trade-off surface. To counteract the effects of this phenomenon and promote diversity, niche induction methods may be used. One such method is that of fitness sharing and is analogous to biological species competing for resources in a natural environment. Individuals that are closer to each other mutually decrease each other's fitness and consequently individuals that are more isolated are given a greater chance of reproducing.

For each individual in the population, a niche count is calculated to determine the degree of crowding around each population member. An individual's niche count is initially set to zero and then increased by a certain amount for every individual in the population including itself. This amount is calculated using a sharing function, which is a function of the distance between two individuals. This distance can be measured in relation to the genotype (the individual population members) or the phenotype (the objective vectors). The niche counts are then used to scale the individuals' fitness values in favour of those that are more isolated.

In this work, fitness sharing is carried out in the objective domain since the aim is to promote diversity with respect to the actual objective values. The following sharing function, often referred to as the triangular sharing function (Goldberg and Richardson, 1987), is used,

$$\begin{aligned} Sh(d) &= 1 - \left(d \middle/ \sigma_{share} \right) && \text{for} \quad d < \sigma_{share} \\ Sh(d) &= 0 && \text{for} \quad d \geq \sigma_{share} \end{aligned} \qquad 6\text{-}7$$

Where d is the distance between population members and σ_{share} is the sharing parameter and dictates how close two individuals must be to each other in order to begin decreasing each other's fitness.

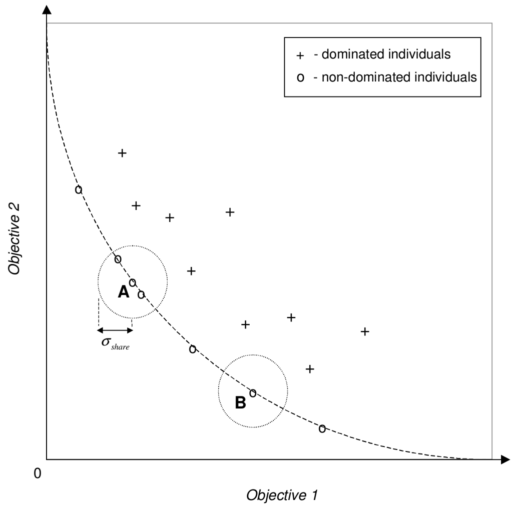


Figure 6-4 – Fitness sharing parameter, σ_{share}

The example shown in Figure 6-4 shows the distance σ_{share} in relation to two non-dominated individuals. It can be seen that there are two individuals within the distance σ_{share} for population member A, while there a no other individuals within the same distance of B. Consequently, the fitness of A will be reduced relative to that of B as it is deemed to be in a more crowded region of the trade-off surface. This will hopefully enable the algorithm to sample the non-dominated front more evenly, as opposed to converging around particular regions of the search space.

Following Fonseca and Fleming, 1998 the distance, *d*, was measured using the ∞-norm. An alternative, and perhaps more obvious method, would be to use the 2-norm. However, since the values of the different objectives are non-commensurable, the 2-norm measure of distance between does not have any significant meaning. The ∞-norm is therefore more appropriate and has the advantage of being easier to compute.

The niche count, m_i for population member *i* is then given by:

$$m_i = \sum_{j=1}^{N} Sh(d_{ij}) \qquad 6\text{-}8$$

Where *N* is the number of individuals and d_{ij} is the distance between individuals *i* and *j*. Fitness sharing is only carried out between sets of equally ranked individuals. This ensures that additional selective pressure is given to the more isolated individuals while still maintaining the ordering imposed by the original ranking process. This technique was first proposed by Horn *et al.* (1994) who referred to it as ‘equivalence class sharing’.

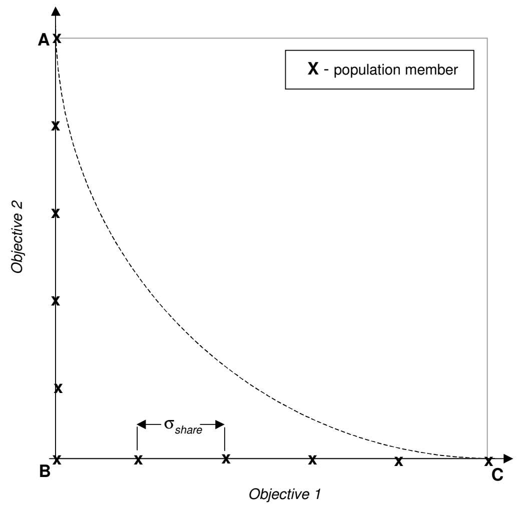


Figure 6-5 – Niche size determination

Since the case studies in this thesis will deal with problems consisting of only two objectives, the method used for determining the sharing parameter σ_{share}, is outlined below with reference to a bi-objective example. Figure 6-5 shows an example Pareto front, $\overrightarrow{AC}$, for two objectives scaled in the range [0 1].

An estimate for the parameter σ_{share} can be calculated by considering the maximum possible length of the Pareto front. It can be seen from Figure 6-5 that, the length of the Pareto front can be no greater than the distance $\overrightarrow{ABC}$. If $\overrightarrow{ABC}$ is populated with N evenly spaced individuals, the distance between each individual is given by,

$$\sigma_{share} = \frac{\overrightarrow{ABC}}{N-1} \tag{6-9}$$

since $\overrightarrow{ABC} = \overrightarrow{AB} + \overrightarrow{BC} = 1 + 1 = 2$

$$\sigma_{share} = \frac{2}{N-1} \tag{6-10}$$

This result is the same as that obtained using the method of Fonseca and Fleming (1998), who proposed a more general approach, applicable to niche size determination in higher dimensional objective space.

Unlike the Pareto-based ranking scheme outlined earlier, the fitness sharing procedure outlined above requires objective values to be scaled in the range [0 1]. The maximum and minimum values used to carry out the scaling must be chosen carefully, so that all objectives are compared equally. Possible values include the known global maximum and minimum or the maximum and minimum discovered so far in the algorithm run. However, these options were thought to be inappropriate for the problems under investigation in this thesis due to RMS error being used as one of the objectives. The RMS errors associated with models generated by a GP algorithm can vary by many orders of magnitude. Although this is most likely to be the case at the start of the algorithm run when the population consists of randomly generated model strings, one cannot guarantee that some highly unfit offspring will be created in subsequent generations, even if a large proportion of the population has a high level of fitness.

Therefore, it was decided to use the maximum and minimum values associated with the equally ranked individuals taking part in fitness sharing. This approach reduces the probability of the scaled RMS values becoming bunched together by the presence of a few very unfit population members.

There are a number of other sharing methods that could be used in conjunction with the MOGP algorithm. For example, the non-dominated sorting genetic algorithm (NSGA) proposed by Srinivas and Deb (1994) uses a quadratic sharing function and requires the number of niches to be specified at the start of the algorithm run in order to determine σ_{share}. There is, however, little evidence to suggest that one method provides substantially better performance. Although comparisons have been performed, they tend to concentrate on relatively simple test problems using GAs, meaning the results may not translate directly to the work in this thesis (see for example, Watson, 1999).

6.2.3.2 Mating Restriction

The use of mating restriction was first proposed by Goldberg (1989) as a method of preventing the production of highly unfit individuals known as lethals. Mating restriction works by only allowing an individual to mate with other individuals that are within a certain distance, σ_{mate}. This approach is intended to prevent the production of highly unfit individuals by preventing mating between individuals that have vastly different objective values.

Although some studies have revealed that improved performance can be achieved by incorporating mating restriction schemes (Hajela and Lin, 1992), others have found no evidence to suggest that such a scheme is necessary and consequently make no use of mating restriction. For example, the strength Pareto evolutionary algorithm (SPEA) outlined by Zitzler and Thiele (1999) makes use of fitness sharing but does not use mating restriction. The MOGP algorithm used in this work does not have a restricted mating scheme.

6.2.3.3 Secondary Populations

The stochastic nature of GP means that there is no guarantee that desirable solutions will be preserved from generation to generation and be present in the population when the algorithm terminates. This is especially important in the case of a MOGP algorithm that may be applied to a problem where the solution set may consist of an extremely large number of non-dominated solutions. In this case, it is impossible for the relatively small population to be able to preserve all non-dominated individuals from one generation to the next while also attempting to exploit new regions of the search space. For this reason, MOEAs often make use of an additional or secondary population in which to store all of the non-dominated or preferable individuals found so far. Some practitioners, for example Zitzler and Thiele (1999), make use of secondary populations that are integrated with the EA and actively take part in the evolutionary process by providing members to take part in generating the next population.

Although there are advantages in using this approach, for example increased diversity may be achieved, a study of such methods was thought to be beyond the scope of this thesis. In addition, the use of a secondary population would make comparison with the SOGP algorithm more difficult as the MOGP algorithm would effectively be working with a larger population size. A simpler method, implemented by the MOGP algorithm used in this work is to simply find the current non-dominated set of individuals at each generation and copy them to the secondary population. The aim of the secondary population is to store a record of all the known non-dominated solutions found so far. As some of the newly added members may dominate some of the existing members, the secondary population will have to be periodically trimmed by ranking the individuals and removing any dominated solutions.

The use of the non-dominated set of individuals taken from every generation of a run is sometimes referred to as the *offline* algorithm performance. In contrast, *online* performance only considers the non-dominated individuals that are present in the population at the end of each run.

6.2.3.4 Summary of MOGP algorithm settings

Figure 6-6 shows a flowchart of the MOGP algorithm used in this thesis. The most significant difference between this MOGP and SOGP algorithms is that the former makes use of a Pareto based ranking scheme with fitness sharing. A secondary population of individuals is also maintained in order to preserve all non-dominated/preferable models along with their objective values. This population is updated at each generation by adding the latest set of preferable individuals. Ranking is then performed in order to remove any individuals that are dominated by the addition of these new population members.

Table 6-1 - MOGP Algorithm details

Model structure	*m* basis functions. *m* generated as a uniformly random integer in range [1 10]
Function set	+, -, /, *, ^, ^2, ^3, `q0`, `q1`, `q2`, `q3`
Terminal set	Process input(s), $u_{1,k-1},...,u_{n,k-1}$ scaled in range [0 1] Model output, $\hat{y}_{k-1}$ $\Re$ generated uniformly in range [-10 10]
Crossover probability	0.7
Mutation probability	0.2
Direct reproduction probability	0.1
Generation gap	90%
Fitness assignment	RMS error + additional objectives $\rightarrow$ Pareto ranking using preference information $\rightarrow$ Fitness sharing

A number of other MOEA techniques have been proposed, making use of alternative ranking, sharing and fitness assignment methods to those described earlier. Examples include the niched Pareto genetic algorithm (NPGA, Horn *et al.*, 1994), NSGA (Srinivas and Deb, 1994) and SPEA (Zitzler and Thiele, 1999). Although Zitzler *et al.* (2000) have shown how these algorithms can outperform Fonseca and Fleming's MOGA, the comparison is restricted to a group of test functions. In addition, each algorithm's performance is assessed in terms of its ability to discover solutions along the entire length of the Pareto optimal front.

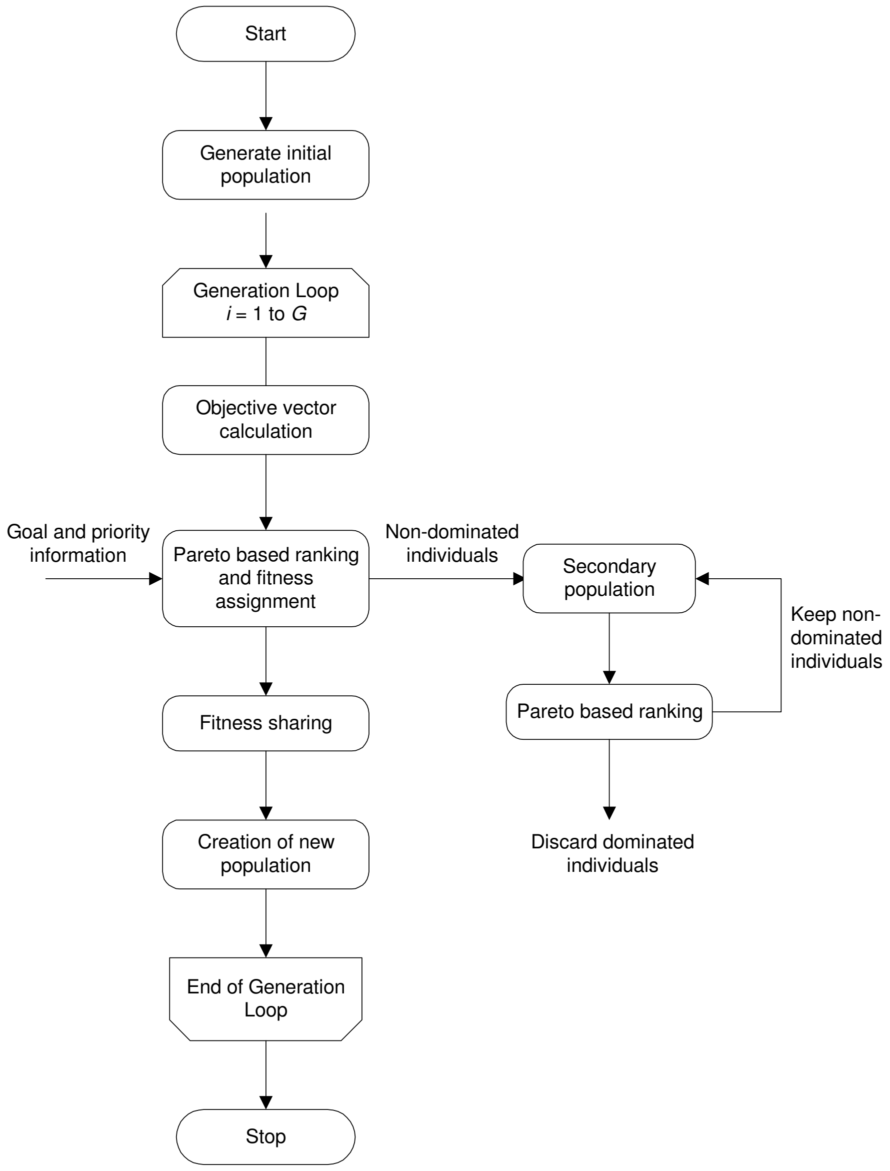

Start
Generate initial population
Generation Loop i = 1 to G
Objective vector calculation
Goal and priority information
Pareto based ranking and fitness assignment
Non-dominated individuals
Secondary population
Keep non-dominated individuals
Pareto based ranking
Discard dominated individuals
Fitness sharing
Creation of new population
End of Generation Loop
Stop

Figure 6-6 – Flowchart for MOGP algorithm

Unfortunately, the objective space associated with real-world engineering problems is much more complicated than that of a simple test function. Consequently, the greatest difficulty is that of being able to direct the search towards the required region of the trade-off surface. As the specification of preference information provides a higher level of control over the algorithm search, this was an important reason for selecting the ranking method used in this thesis. The important settings and features of the MOGP algorithm are outline in Table 6-1.

7 Dynamic Modelling Using Multi-objective GP

7.1 Introduction

The previous chapter outlined the modifications necessary to apply the MBF-GP algorithm to multi-objective problems. This chapter demonstrates the application of the algorithm to dynamic model development. Since the notion of algorithm performance is more complex than for the single objective problem, the first part of this chapter describes the methods used to compare MOGP algorithm results. The algorithm is then applied to two problems, each involving the incorporation of one additional measure of model performance. The first example uses the MOGP algorithm to improve the parsimony of the evolved model structures. The second example demonstrates how residual correlation tests can be used as an additional objective. In each case, the algorithm is applied to an artificial test system and a process engineering case study.

7.1.1 Measures of MOEA Performance

The performance of a MOGP algorithm is more difficult to quantify than the single objective case. For example, Zitzler *et al.* (2000) outlined the following criteria for measuring the performance of MOEA runs,

- The distance of the non-dominated solutions from the known Pareto optimal set should be minimised.
- A uniform distribution of the solutions over the front is desirable.
- The extent of the non-dominated front should be maximised, so that the solutions cover a wide range of values in each objective domain.

However, these criteria were used to compare the performance of various MOGA implementations when applied to a set of synthetic test functions and there are a several reasons why they may be unsuitable for real-world applications. Firstly, in this work the objective values for the global Pareto optimal solutions are not known, meaning that it is impossible to measure the distance of the evolved Pareto front from the actual solutions. Secondly, test studies often compare the range of values covered by the Pareto optimal set to measure the algorithms ability to find a diverse set of candidate solutions. This may not necessarily be the best approach for a complex engineering problem as only a small region of the trade-off surface may contain solutions that provide a useful compromise between the different performance criteria. For example, an engineer may require a model that is both parsimonious *and* accurate and will not want the algorithm to discover solutions that perform very well in only one of the criteria.

Zitzler *et al*. (2000) made a comparative study of the most popular MOGA implementations found in the literature. The results ranked the method of Fonseca and Fleming (FFGA) behind the Pareto based algorithms SPEA, NSGA, NPGA and even the non-Pareto methods of VEGA and HLGA. However, as mentioned previously, the main reason for choosing a method based on FFGA is its successful application to real engineering problems where the aim is to use preference information to guide the algorithm towards the relevant part of a complex search space. In addition, in MOGA performance studies, the GA is concerned solely with the evolution of the parameters belonging to a fixed functional relationship. The problem is more complicated for the MOGP case, as the algorithm is also responsible for evolving the model structure. This is further justification for concentrating the search towards the desired region of the non-dominated front instead of stretching algorithm resources by trying to discover solutions along the entire length of the front. The method used to compare algorithm performance in this thesis is described in the next section.

7.1.2 Analysis procedure

Although the SOGP algorithm only uses RMS error to measure the fitness of each population member, additional measures of model performance can be calculated for the final population at the end of the algorithm run. These objective values can be used in conjunction with Pareto ranking to find a non-dominated set of solutions describing the trade-off between the different objectives. These solutions can then be compared with those obtained using the MOGP algorithm to determine the benefits of using multi-objective techniques during model evolution.

The first stage of the experimental procedure is to compare the MOGP algorithm using conventional Pareto ranking with the SOGP algorithm. Comparison is made by comparing the non-dominated set of individuals obtained from a set of twenty algorithm runs. This set is found by combining the non-dominated solutions from each run and then re-ranking the individuals to find the non-dominated members. The non-dominated individual's objective values can then be plotted to illustrate the trade-off between the different measures of algorithm performance.

One of the most common techniques for comparing non-dominated fronts produced by MOEAs is to use visual inspection. Additional approaches have been proposed that use specially designed metrics to quantify the differences between certain aspects of the non-dominated front. One of the drawbacks of these methods is that it is extremely difficult to define a single metric that allows all of the desired performance criteria to be combined in a meaningful way. The results of such analyses have to be interpreted with care as each performance measure has its advantages and disadvantages. For example, Zitzler *et al.* (2000) describe a metric used to calculate the fraction of individuals produced by one algorithm that dominate those generated using another algorithm. One of the disadvantages of this technique is that it does not consider the magnitude of the difference between the objective values. In contrast, with visual inspection it is easy to observe whether one front dominates another by a very large or small margin. In this thesis, no performance metrics have been used to characterise the non-dominated fronts produced by different algorithms and all observations were made by visual examination.

As some regions of the objective space (and consequently the non-dominated front) may be sampled more densely, histograms are used to compare the distribution of the final population members' objective values. These distributions along with the non-dominated fronts provide information that can be used to determine how preference information can be used to enhance algorithm performance. For example, the distribution of the final population members may indicate that an algorithm has tended to bias its search in favour of one of the objectives. Additional sets of runs can then be carried out with preference information and the results compared to those achieved using Pareto ranking.

7.2 Model parsimony

The results in previous chapters showed that, although GP is able to develop accurate models of steady-state and dynamic processes, the resulting structures are rather complex. Even for simple test systems, the MBF-GP model expressions contained a relatively large number of basis functions. Although the models produced accurate predictions when applied to the validation data, parsimonious representations are more desirable as it is generally thought that they are more likely to generalise well. Complex models may include superfluous model terms that lead to overfitting of the training data resulting in poor performance when applied to unseen data. During any model development process, a balance must be reached between model accuracy and complexity, as there is little benefit in greatly increasing model complexity if the relative improvement in accuracy is not significant. From a practical point of view, it is preferable to work with a parsimonious model structure rather than a highly complex representation with the same prediction accuracy. In addition, a population of unnecessarily large model expressions will require a greater amount of processing time and reduce the efficiency of the algorithm.

Conventional model development procedures often result in a set of candidate models with varying levels of accuracy and complexity. One way to select the most appropriate solution is to apply statistical information criteria such as the Akaike information criterion (AIC) and the final prediction error (FPE), which attempt to strike a balance between the prediction accuracy and complexity of the model. The AIC and FPE for a particular model are given by the following relationships,

$$AIC = N\ln\left[\frac{1}{N}\sum_{k=1}^{N}\varepsilon(k)^2\right] + 2\varphi \qquad 7\text{-}1$$

$$FPE = N\ln\left[\frac{1}{N}\sum_{k=1}^{N}\varepsilon(k)^2\right] + N\ln\left[\frac{N+\varphi}{N-\varphi}\right] \qquad 7\text{-}2$$

Where $\varepsilon(k) = y(k) - \hat{y}(k)$ is the residual at time sample *k*, $y(k)$ is the measurement, $\hat{y}(k)$ is the model prediction, φ is the total number of terms included in the model and *N* is the number of data points. It can be seen from equations 7-1 and 7-2 that both criteria consist of two terms, one for prediction error and one for model order. The first term is a measure of how well the model fits the data and the second is a measure of the complexity of the model required to achieve the fit. AIC and FPE therefore attempt to find a compromise between low residual variance and an excessive number of model parameters with smaller values indicating more desirable model structures.

Ideally, if AIC is statistically consistent, it will obtain a minimum value for the correct number of model parameters. However, it can be demonstrated that the AIC is statistically inconsistent and tends to overestimate the model order (Johansson, 1993). Similar problems can be encountered when using FPE, which has been found to underestimate the correct order of the system. Consequently, a number of other criteria have been proposed, such as the Bayesian information criterion (BIC), law of iterated logarithms criterion (LILC) and the minimum description length (MDL) principle. Iba *et al.* (1994) proposed a MDL based fitness function in order to control the size of the models produced by their GP algorithm (STROGANOFF).

A disadvantage of using such criteria in conjunction with a GP algorithm is that GP models can be very complex yet may not necessarily have a large number of numerical parameters. Consequently, the algorithm may evolve models that have relatively low AIC values by generating models consisting of a few extremely complex basis functions. This will defeat the overall aim of evolving parsimonious model structures. This is not an issue with STROGANOFF as the functional nodes are restricted to linear polynomials, each with a fixed number of parameters.

It is well documented in the GP literature that program trees expand as the evolutionary process proceeds. This process, known as 'bloat', has been widely reported by a number of researchers, and it is common that such increases in program size occur without any significant improvement in performance (Koza, 1992, Blickle and Thiele, 1994, Nordin and Banzhaf, 1995). Consequently, the performance of the

GP algorithm may be compromised if model trees expand to the maximum allowable size

It has been suggested that code growth occurs in GP algorithms as a means of protecting solutions from destructive crossover operations (Blickle and Thiele, 1994, Nordin and Banzhaf, 1995). This is due to the fact that large GP programs will often contain sections of code that serve no useful purpose and when executed do not contribute to the individual's fitness. These sections, known as introns, may help to prevent the creation of unfit solutions by providing sites for crossover to take place non-destructively. Nordin *et al*. (1996) used a symbolic regression problem to demonstrate how the inclusion of explicitly defined introns (EDIs) could lead to improved generalisation and reduced processing time.

Another method used by GP practitioners to evolve parsimonious solutions is to directly limit the growth of the program trees. This can be done by setting a maximum program size that must not be exceeded by new offspring if they are to be allowed to take part in the next generation. However, this approach is not very flexible, as the maximum program size must be set at the start of the run. If unsuccessful results are obtained, the program size will have to be increased and more algorithm runs performed until a successful solution is generated.

Another commonly used way of implementing parsimony pressure is to use a fitness term that incorporates a penalty function based on program size. The fitness of individual *i* is then given by,

$$F_i = P_i - p(s_i) \tag{7-3}$$

Where P_i is a measure of how individual *i* actually performs, s_i is the size of *i* and *p* is a function used to apply parsimony pressure. If *p* is chosen to be a simple linear function, the fitness function becomes,

$$F_i = P_i - \beta s_i \tag{7-4}$$

Where β is a weighting parameter used to adjust the strength of the parsimony pressure to be applied. Although some researchers have reported successful applications of parsimony pressure (Soule *et al.*, 1996, Blickle, 1996), others have experienced less favourable results (Koza, 1992, Nordin and Banzhaf 1995), reporting that increased parsimony lead to a reduction in the performance of the evolved solutions. A possible disadvantage is that it may be difficult to find the value of β in equation 7-4 that enables the algorithm to evolve parsimonious solutions without degrading their performance.

The aim of this work is to investigate whether the MOGP algorithm can provide a more flexible approach. By applying the concepts of Pareto dominance, the algorithm can be used to generate a set of candidate solutions from which to choose the final model. This means that fewer decisions have to be made (e.g. setting cost function weightings) before the end of the algorithm run, thus avoiding some of the drawbacks of other methods. Preference information can be used to enhance performance by directing the search towards the desired region of the non-dominated front.

It has been shown (Hinchliffe *et al.*, 1998) how the MOGP algorithm described in the previous chapter can be used to generate parsimonious models of steady-state processes without compromising model accuracy. Rodríguez-Vázquez and Fleming (1998) used a number of model attributes such as the maximum process lag and polynomial order when evolving NARX models using a similar algorithm. However, if GP is to be used to automatically develop dynamic models, it must be assumed that aspects of the final model structure such as the maximum process lag are unknown. It is therefore difficult to specify goals for such objectives before the algorithm run commences. This was not a disadvantage for Rodríguez-Vázquez and Fleming as their approach allows preference information to be adjusted during model evolution.

Following Hinchliffe *et al.* (1998), the string length of a GP model expression is used in this thesis to measure model parsimony. This value is easy to calculate and does not require a significant amount of additional processing. Another advantage is that the GP algorithm is still responsible for model characteristics such as the number of lags and degree of non-linearity as no restrictions (i.e. preference information) have to be placed on these criteria.

7.2.1 Analysis of Results

7.2.1.1 Test System with Time Delay

Initial algorithm runs were performed to compare the SOGP algorithm with the MOGP algorithm using Pareto ranking (MOGP-P). An additional set of MOGP-P runs was carried out without fitness sharing (FS) to demonstrate its effect on algorithm performance. Each algorithm was run 20 times with a population size of 50 for 50 generations. The non-dominated sets of individuals obtained from each set of runs are compared in Figure 7-1 (RMS errors are on the training data set).

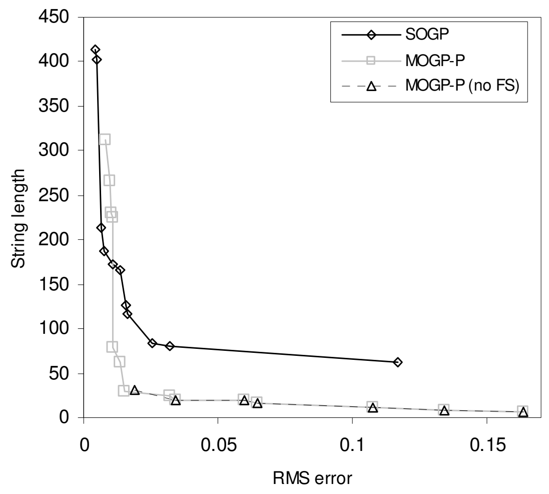
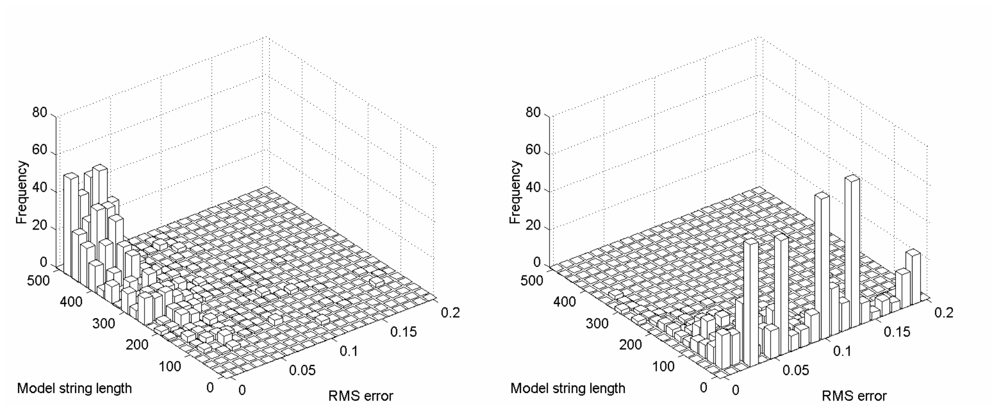


Figure 7-1- Comparison of non-dominated fronts achieved by SOGP and MOGP-P algorithms

The non-dominated fronts contain sets of solutions that represent a trade-off in performance between the two objectives. The solutions range from parsimonious models with high RMS errors to increasingly complex solutions with lower prediction errors. Figure 7-1 shows that the MOGP-P algorithm evolved the more parsimonious solutions with 8 of the 14 non-dominated individuals having a string length of fewer than 50 characters. Unfortunately, the reduction in complexity has been achieved at the expense of accuracy, with the most accurate of these models having an RMS of 0.0152 compared to the most accurate SOGP model, which has an RMS of 0.00429. As the accuracy of the MOGP-P models improve, there is a sharp increase in model complexity. The MOGP-P model with the lowest prediction error has an RMS error of 0.00839. Without fitness sharing, the MOGP-P algorithm can only manage to generate a set of relatively inaccurate models that are all under 50 characters in length.

Although the SOGP and MOGP-P algorithms have evolved non-dominated sets of solutions covering a wide range of objective values, the distributions of the final population members must be studied to examine how evenly the individuals are scattered along the non-dominated front. The distributions of the final population members' objective values for the SOGP and MOGP-P algorithms are shown in Figure 7-2. Figure 7-2a shows how the majority of the individuals produced by the SOGP algorithm have expanded towards the maximum permissible string length of 500 characters. The resulting models provide accurate predictions but are rather complex.

The effect of including the string length objective during model evolution is demonstrated by Figure 7-2b. The MOGP-P algorithm has produced a large number of parsimonious model structures, with only a fraction having RMS errors as low as the models produced by the SOGP algorithm. This is probably because it is easier for the algorithm to generate parsimonious models than it is to evolve models that give accurate predictions. Whereas the algorithm may require a considerable number of generations to evolve models with low RMS errors, parsimonious models will be present in the initial population. These models will be selected to take part in reproduction due to their high level of performance in one objective dimension. The action of genetic operators on these models is more likely to produce parsimonious solutions instead of ones that have low RMS errors. As the population becomes saturated with simple model structures, it will become increasingly difficult for the algorithm to evolve accurate models via crossover and mutation. Although the fitness sharing scheme is intended to promote diversity, it is unable (in this case) to prevent the population members from becoming unevenly distributed along the non-dominated front.



(a) Single objective (b) MOGP-P algorithm

Figure 7-2 – Distributions of final population members' objective values

Figure 7-3 shows that the bias towards the string length objective is more pronounced when fitness sharing is not included. In this case, the algorithm produces an extremely large number of solutions with very short string lengths and relatively high RMS error values. The lack of a fitness sharing scheme means that additional fitness is not allocated to diverse individuals (in this case, more accurate but more complex solutions). Consequently, the population becomes saturated with parsimonious solutions that perform poorly in terms of RMS error. Without sharing, the tendency of the algorithm to favour model parsimony is so exaggerated that virtually all of the models are constructed from a single input terminal.

Although the MOGP-P algorithm has discovered a large number of parsimonious solutions, model accuracy has been compromised. This problem can be addressed by using preference information to guide the search towards the desired region of the non-dominated front. The MOGP-P algorithm can be prevented from biasing its search in favour of string length minimisation by setting a goal value for that objective. The preferability relationship outlined in section 6.2.2.1 selectively omits objectives from the ranking process once they have satisfied their goals. Consequently, goals can be used to prevent the algorithm from attempting to minimise an objective that has already reached a desirable level of performance.

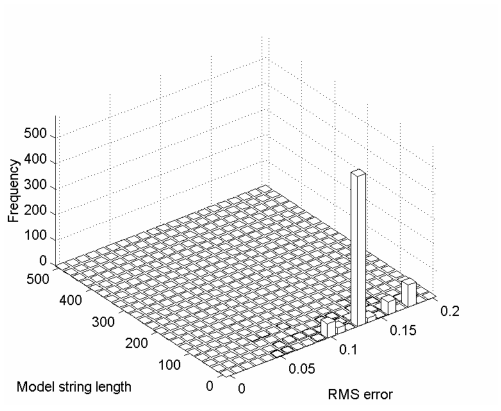


Figure 7-3 –Distribution of objective values for MOGP-P algorithm without fitness sharing

An additional set of runs was performed with the MOGP algorithm using Pareto ranking with preference information (this algorithm will be referred to as MOGP-FF due to its use of Fonseca and Fleming's preferability relation). The RMS error goal was set to an 'ideal' value of zero and the model string length goal was arbitrarily set to 100 characters. This means that population members with string lengths below this value are compared using their RMS values alone. With this approach, the algorithm will not try to minimise the string length below 100 characters and avoids the evolution of overly parsimonious solutions. Figure 7-4 compares the non-dominated front obtained using the MOGP-FF algorithm with the fronts achieved by the SOGP and MOGP-P algorithms.

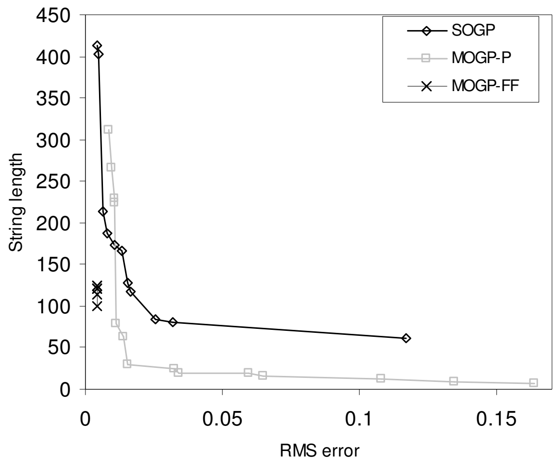


Figure 7-4 – Comparison of non-dominated fronts

The plots show how the non-dominated solutions evolved by the MOGP-FF algorithm are grouped closely around the 100 character goal value. The corresponding RMS error values are lower than those achieved by all of the MOGP-P solutions. The most accurate model produced by the MOGP-FF algorithm has an RMS value of 0.00433 compared to a value of 0.00429 for the most accurate SOGP model. The corresponding string length is considerably shorter (125 compared to 414 characters), meaning that the MOGP-FF algorithm has been able to reduce model complexity without a significant loss of prediction accuracy. The distribution of the individuals from the final populations of the MOGP-FF runs is shown in Figure 7-5. The inclusion of preference information has enabled the algorithm to develop accurate and parsimonious process models, with the greatest concentration of models located around the 100 character goal value. This is in contrast to the SOGP results, which demonstrate how 'bloat' leads to the generation of solutions that occupy the maximum allowable program size.

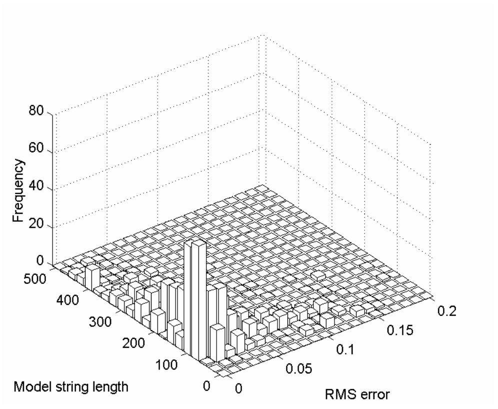


Figure 7-5 - Distribution of objective values for MOGP-FF algorithm

The non-dominated set of solutions provides the engineer with a trade-off between model parsimony and accuracy. Before the final model is chosen, the validation RMS of each model must also be considered. Table 7-1 shows the validation RMS values of the six non-dominated solutions generated using the MOGP-FF algorithm along with their training RMS errors and model string lengths.

Table 7-1- Comparison of non-dominated solutions

Model no.	RMS	String length	Validation RMS
1	0.0043296	125	0.007287
2	0.0043297	124	0.007287
3	0.0043333	121	0.007293
4	0.0043465	120	0.007312
5	0.0043959	114	0.006619
6	0.0044174	100	0.008632

The fifth model in Table 7-1 has the lowest RMS error on the validation data set. The model is constructed from a single basis function and is shown in simplified form below,

$$\hat{y}_k = 0.5114(\hat{y}_{k-1} + 5.512\hat{y}_{k-9}u_{k-11} + u_{k-11} + 0.894) - 0.0451 \qquad 7\text{-}5$$

The model is ranked second in terms of parsimony and probably provides the best overall solution to the problem (in terms of a compromise between the chosen performance criteria). The lowest validation RMS achieved by the SOGP algorithm is slightly lower (0.00606 compared to 0.00662) but the model is more complex, containing 13 basis functions.

7.2.1.2 Cooking extruder

As in the previous section, SOGP, MOGP-P and MOGP-FF algorithm runs were carried out in order to compare algorithm performance. Twenty runs of each algorithm were performed using a population size of 50 individuals for 50 generations. As before, the MOGP-FF algorithm RMS and string length goals were set to values of 0 and 100 respectively. The non-dominated fronts obtained are shown in Figure 7-6

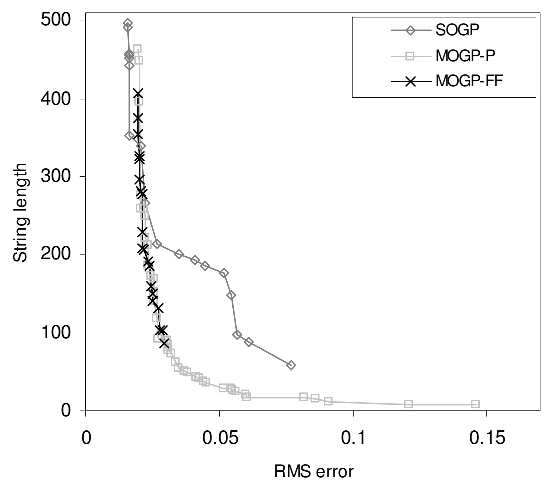
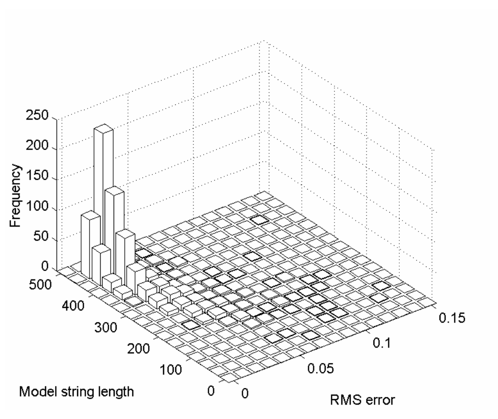


Figure 7-6 – Comparison of non-dominated fronts achieved by SOGP and MOGP algorithms

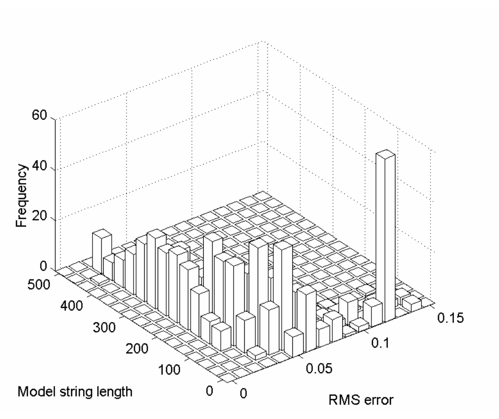
The non-dominated set of solutions evolved by the MOGP-P algorithm covers a wide range of objective values. The most accurate MOGP-P model has an RMS error of 0.0197 and a string length of 463 characters. At the opposite end of the front, the most parsimonious solution is only seven characters long and has an RMS of 0.146. Although the SOGP algorithm was able to evolve a number of relatively simple solutions, models of fewer than 200 characters in length have substantially higher RMS errors than MOGP-P models of the same size. The SOGP algorithm evolved the model with the lowest training RMS (0.0158 compared to 0.0197 for the MOGP-P algorithm). This would be expected, as all SOGP algorithm resources are concentrated on the task of evolving models that perform well in terms of this objective.

The distributions of the end-of-run objective values are compared in Figure 7-7. Figure 7-7a demonstrates how the SOGP models have expanded to fill the maximum available string size of 500 characters. Figure 7-7b shows that there is a relatively even distribution of individuals along the non-dominated front evolved by the MOGP-P algorithm. This is in contrast to the distribution that is achieved by the same algorithm without fitness sharing (Figure 7-7c). It can also be seen that the MOGP-P algorithm has produced a large cluster of individuals that have short string lengths and

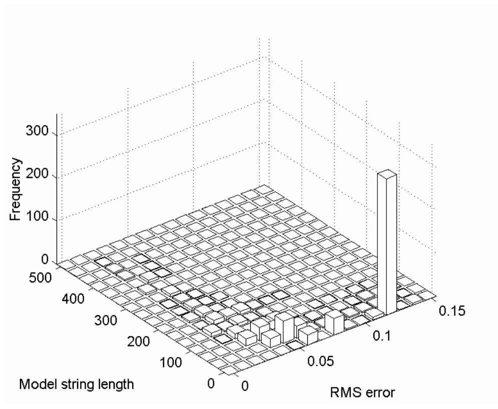
RMS error values of approximately 0.12. The bias towards the string length objective was much greater for the test system studied in the previous section.



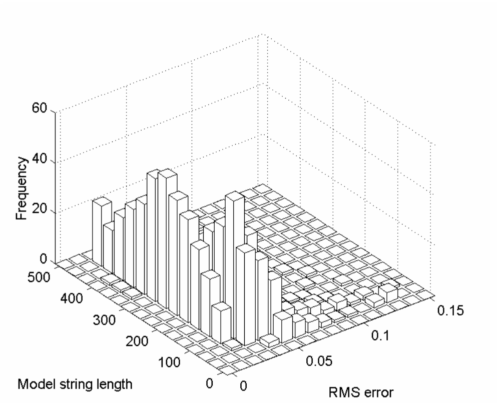
(a) Single objective



(b) MOGP-P algorithm



(c) MOGP-P algorithm without fitness sharing



(d) MOGP-FF algorithm

Figure 7-7 – Distribution of objective values for final generation population members

The non-dominated set of solutions obtained using the MOGP-FF algorithm does not contain the highly parsimonious but inaccurate models that are found in the MOGP-P set. In the test system case study, the MOGP-FF algorithm was able to evolve models that were more parsimonious and had lower RMS errors than the other algorithms. This is not the case in this example, as the non-dominated front evolved by the MOGP-FF algorithm closely follows that evolved by the MOGP-P algorithm.

However, Figure 7-7d shows that the MOGP-FF produced fewer models with extremely short string lengths and high RMS errors, indicating an improvement in terms of the consistency of individual algorithm runs.

Table 7-2 – MOGP model with lowest validation RMS

Basis function	Parameter
$\hat{g}_{k-4}$	0.29736
ω_{k-2}	0.49312
$M^3_{f\,k-2}$	-0.039973
$Q_{f\,k-4}$	-0.27842
ω_{k-4}	0.054084
$T^2_{f\,k-4}$	-0.073058
$Q^4_{f\,k-3} - 0.57363$	0.18924
Bias	0.1441

The validation RMS must also be considered before the final model structure can be selected. The SOGP model with the best performance when applied to the validation data had a validation RMS error of 0.0399 and a string length of 492 characters. The best MOGP-FF model had a slightly lower validation RMS error of 0.0385 and a string length of only 131 characters. The MOGP model contains seven parsimonious basis functions, compared to the sixteen basis functions of the SOGP model. The MOGP model is shown in Table 7-2 in simplified form.

7.2.2 Discussion

Both case studies showed how the MOGP-P algorithm was able to generate a set of non-dominated solutions that possessed varied levels of performance in each objective. The main disadvantage of this technique is that many of the solutions do not have a high enough level of performance with respect to both objectives. In this example, the algorithm tended to generate a large number of models that had short string lengths but higher RMS errors than the SOGP algorithm. The MOGP-P results are useful, however, as they highlight the regions of objective space that are being over exploited by the algorithm. This information can then be used to help select suitable goal values to be supplied to the MOGP-FF algorithm.

This MOGP-FF runs illustrated how preference information can be used to concentrate the search towards the desired region of the objective space. The benefits of using the MOGP-FF algorithm were greater for the test system, where the algorithm was able to generate concise solutions that provided more accurate predictions than the SOGP and MOGP-P algorithms. A smaller improvement was seen with the extruder data, but this would be expected for a more complex system. The work carried out in this section also emphasised the importance of the fitness sharing scheme. Without fitness sharing, all of the MOGP-P solutions converged closely around round the same region of the search space. This was undesirable, as the solutions were overly parsimonious and produced very poor predictions.

7.3 Residual analysis

The residuals of a model represent the difference between the predicted and actual values of the process output. Consequently, the presence of any information remaining in the residuals is an indication that the proposed model may be inadequate in some way. The existence of such information can be investigated by using a number of techniques that take into account factors such as (Johansson, 1993),

- Correlations between residuals and the inputs(s) or output.
- The autocorrelation of the residuals
- Normal distribution of residuals
- Zero crossings (changes of sign) of the residual sequence

These techniques are usually applied after the model structure and associated parameters have been identified. A potential advantage of using a MOGP approach is that performance with respect to the tests could be taken into account throughout the model evolution process.

Although any combination of these criteria could be used as objectives within a MOGP framework, the correlation tests outlined by Billings and Voon (1986b) are used in this thesis. The tests were designed to determine whether a proposed model captures the dynamics of the underlying process as opposed to simply fitting the available data. These tests have been applied to a number of non-linear dynamic systems including real and simulated processes (Doherty *et al.*, 1997, Arkov *et al.*, 2000). In addition, it has been shown how the test results can be used to identify process lags that are missing from the input sequence. Similarly, providing the GP algorithm with the same measure of model performance may increase the algorithm's ability to discover the correct combinations of inputs and process lags required to evolve an accurate model.

For linear systems, the auto-correlation function of the residuals and the cross-correlation between the residuals and the input(s) are sufficient model validation tests (Söderström and Stoica, 1989). However, for non-linear systems, additional higher order tests are required to detect the presence of unmodelled linear and non-linear terms. In total, the following five tests have been proposed (Billings and Voon, 1986b),

$$\begin{array}{lll} \text{(a)} & \phi_{u\varepsilon}(\tau) = 0 & \forall \tau \\ \text{(b)} & \phi_{\varepsilon\varepsilon}(\tau) = \delta(\tau) & \forall \tau \\ \text{(c)} & \phi_{\varepsilon\varepsilon u}(\tau) = 0 & \forall \tau \\ \text{(d)} & \phi_{(u^2)'\varepsilon}(\tau) = 0 & \forall \tau \\ \text{(e)} & \phi_{(u^2)'\varepsilon^2}(\tau) = 0 & \forall \tau \end{array} \qquad 7\text{-}6$$

Where τ is the time-shift and δ is the Kronecker delta function. The necessary expressions for obtaining the correlation estimates are as follows:

$$\phi_{xy}(\tau) = \frac{\sum_{k=\tau}^{N} (x(k) - \bar{x})(y(k-\tau) - \bar{y})}{\sqrt{\sum_{k=1}^{N} (x(k) - \bar{x})^2 \sum_{k=1}^{N} (y(k) - \bar{y})^2}} \qquad 7\text{-}7$$

$$\phi_{\varepsilon\varepsilon u}(\tau) = \frac{\sqrt{N}\sum_{k=\tau}^{N}\varepsilon(k)\varepsilon(k-1)(u(k-\tau)-\bar{u})}{\sum_{k=1}^{N}\varepsilon^2(k)\sqrt{\sum_{k=1}^{N}(u(k)-\bar{u})^2}} \qquad 7\text{-}8$$

The first two tests are the standard autocorrelation and cross correlation functions used in linear system identification. The remaining higher order tests are designed to detect missing non-linear terms by examining the correlations between odd and even powers of the inputs and residuals.

Rodríguez-Vázquez and Fleming (1998) used the linear autocorrelation and correlation tests as objectives during the evolution of NARMAX models using a MOGP algorithm. Later, Arkov *et al.* (2000) extended the work by including the additional tests for non-linear systems. Although the authors describe how the technique can be used to develop accurate models, the results were not compared with algorithms that did not use multi-objective techniques. In contrast, the aim of this work is to emphasise the advantages of using the MOGP-FF algorithm in place of the MOGP-P and SOGP algorithms.

The correlation tests are usually performed at the 95% confidence level. This means that the residuals will contain no linear or non-linear structure if the absolute value of each test statistic is not greater than $1.96/\sqrt{N}$. The correlation test objective value (Φ) for a MOGP model can then be found by taking the sum of the test values that exceed the 95% confidence limit,

$$\Phi = \sum_{i=1}^{5}\sum_{j=0}^{\tau_{max}}\left(\left|\phi_i(j)\right| - 1.96/\sqrt{N}\right) \qquad j \neq 0 \text{ if } i=1 \qquad 7\text{-}9$$

Where $\phi_1 = \phi_{\varepsilon\varepsilon}$ $\quad \phi_4 = \phi_{(u^2)'\varepsilon}$

$\phi_2 = \phi_{u\varepsilon}$ $\quad \phi_5 = \phi_{(u^2)'\varepsilon^2}$

$\phi_3 = \phi_{\varepsilon\varepsilon u}$ $\quad \tau_{max}$ = Maximum time shift

For a process with multiple inputs, the test values for each input can be combined to produce a single objective value,

$$\Phi_{total} = \sum_{i=1}^{n} \Phi_i \qquad 7\text{-}10$$

Where *n* is the number of process inputs. Combining 7-10 with equation 7-9 yields,

$$\Phi_{total} = \sum_{j=1}^{\tau_{max}} \left(|\phi_1(j)| - 1.96/\sqrt{N} \right) + \sum_{k=1}^{n} \sum_{i=2}^{5} \sum_{j=0}^{\tau_{max}} \left(|\phi_{i,u_k}(j)| - 1.96/\sqrt{N} \right) \qquad 7\text{-}11$$

Where ϕ_{i,u_k} refers to correlation test *i* using the *k*th process input. Note that the first test is based on the residuals only and does not have to be evaluated for each process input.

If all of the correlation values for time shifts in the range [0 τ_{max}] fall within the 95% confidence region, equation 7-11 produces an objective value of zero. This is convenient as the required performance level for this objective will always be the same, regardless of the maximum lag and number inputs. Another approach would be to use the individual tests as separate objectives. An advantage of this method is that more information is given to the MOGP algorithm in terms of which correlation tests have been satisfied. However, the resulting problem will have a relatively large number of objectives, especially if processes with more than a single input are considered. This may lead to a large number of non-dominated solutions and larger population sizes may be required to enable the algorithm to carry out an effective search. Another reason for combining the tests into a single objective value is that the overall problem has only two objectives, making the trade-off between the objective values of candidate solutions easier to visualise.

The maximum time-shift, τ_{max}, was set to twenty process lags for all of the correlation tests carried out in this work. This is a commonly used value (Doherty *et al.*, 1997, Arkov *et al.*, 2000) and exceeds the maximum time delay encountered during both of the case studies used in this work. In practice, the results of the tests must be analysed carefully to ensure that τ_{max} is not less than the maximum lag required for an accurate model.

7.3.1 Analysis of Results

7.3.1.1 Test System with time delay

Initially, two sets of runs were performed to compare the performance of the MOGP algorithm using Pareto ranking with that of the SOGP algorithm. As with the model string length objective, correlation tests can be performed at the end of each SOGP algorithm run and the values used to rank the population members and find a non-dominated set of solutions to be compared with the MOGP results

Preliminary results indicated that, after 50 generations of the MOGP algorithm run, few models had correlation test values close to zero and consequently all sets of runs were allowed to run for 100 generations. Twenty runs of each algorithm were performed with a population size of 50. Figure 7-8 compares the non-dominated solutions obtained by both algorithms.

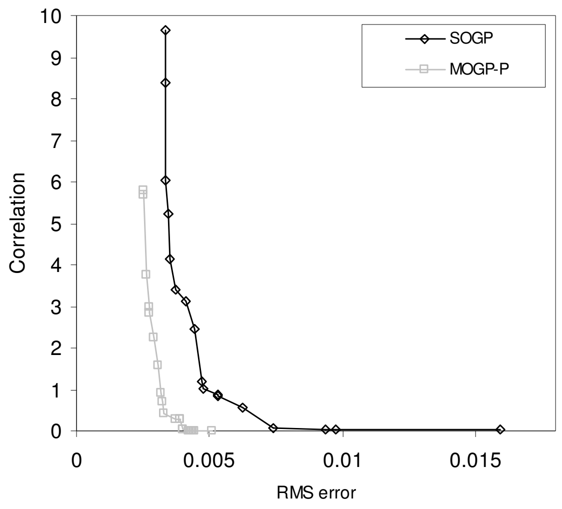
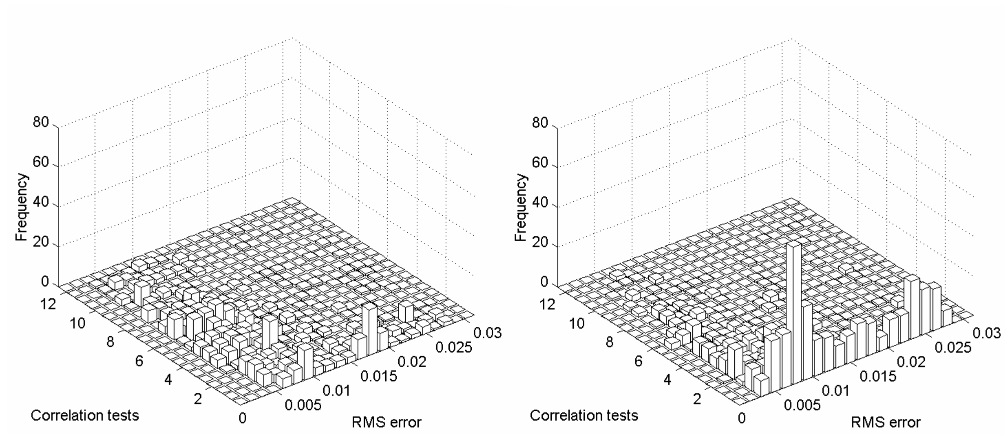


Figure 7-8- Non-dominated fronts obtained using SOGP and MOGP-P algorithms

Figure 7-8 demonstrates the advantage of including the correlation test objective during model evolution. The MOGP-P algorithm has improved performance with respect to both objectives. The most accurate model obtained using the MOGP algorithm has a lower RMS (0.00250 compared with 0.0336) and a lower correlation test value (5.798 compare with 9.665). At the opposite end of the non-dominated front, there are individuals with better performance in terms of correlation test values but degraded performance in terms of RMS error. The MOGP algorithm has outperformed the SOGP algorithm as the model with the best correlation test value for the MOGP algorithm has an RMS error of 0.0508 and a correlation test value of zero compared to values of 0.1591 and 0.0232 for the SOGP algorithm. It can be seen, for any given RMS error, the corresponding correlation test value is lower for the MOGP algorithm and vice versa.



(a) Single objective (b) MOGP-P algorithm

Figure 7-9 –Distribution of final generation objective values for SOGP and MOGP-P algorithms

Despite this improvement, preference information can still be used to evolve a greater number of solutions with increased performance with respect to both objectives. Figure 7-9 reveals that the MOGP-P algorithm appears to have biased its search in favour of the correlation tests, with a substantial number of individuals performing well at the tests but having higher RMS values in the range [0.01 0.03]. This means that although the non-dominated front shows improved performance, individual runs may fail due to their convergence around regions with RMS errors in this range.

A possible solution is to only include the correlation tests when prediction errors have reached a certain level of accuracy (for example an RMS error of 0.01). This can be accomplished by setting goals of 0.01 and ∞ for the RMS and correlation objectives respectively. Setting a value of ∞ means that all population members will satisfy the goal for that particular objective. Consequently, the correlation tests are omitted from the ranking process and individuals are compared using their RMS values only. This will be the case until individuals are encountered that have RMS errors that satisfy the RMS goal (0.01). At this stage, all goal values will be satisfied and the corresponding individuals will be compared in the conventional Pareto manner, meaning that both objectives are taken into consideration. An additional set of twenty runs was performed using the MOGP-FF algorithm with the goal values described above. The non-dominated front achieved by the algorithm is shown in Figure 7-10 compared to the fronts evolved by the SOGP and MOGP-P algorithms.

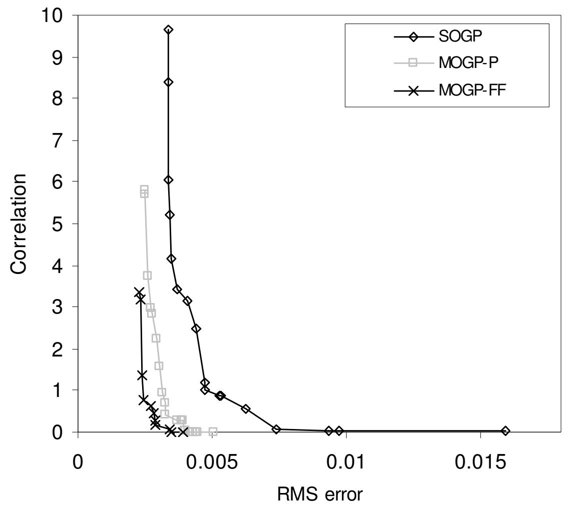


Figure 7-10 – Comparison of non-dominated fronts achieved using SOGP, MOGP-P and MOGP-FF algorithms

Figure 7-10 illustrates how the MOGP-FF algorithm was able to evolve a set of solutions with improved performance with respect to both objectives. For example, the most accurate MOGP-FF model has an RMS error of 0.00231 and a correlation test value of 3.371 compared to values of 0.00250 and 5.798 for the corresponding MOGP-P model. At the opposite end of the non-dominated front, the MOGP-FF and MOGP-P algorithms have both discovered models that have correlation values equal to zero. However, the MOGP-FF algorithm gives a more accurate prediction, with an RMS of 0.00394 compared to a value of 0.00508 for the MOGP-P algorithm. The distribution of the objective values taken from the final generation of each run is shown in Figure 7-11. The addition of preference information has prevented the algorithm from evolving models with low correlation scores and RMS errors greater than 0.01. Without these solutions occupying algorithm resources, more solutions have been evolved with RMS errors in the range [0.005 0.01].

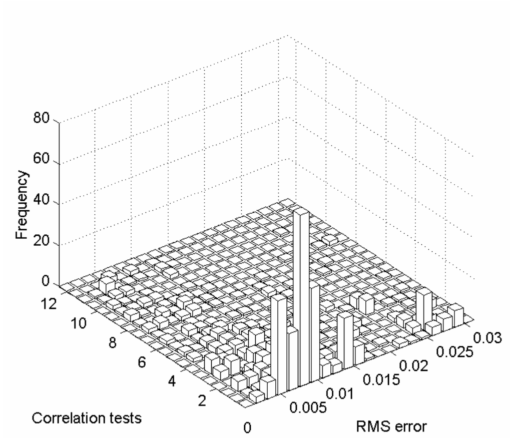


Figure 7-11 – Distribution of final population objective values obtained using the MOGP-FF algorithm

Before the ‘best’ model of these preferable models can be chosen, the performance of the models on the validation data must also be taken into consideration. Table 7-3 contains the objective values and validation RMS errors belonging to the non-dominated solutions evolved using the MOGP-FF algorithm. Table 7-3 shows how low RMS errors can be obtained on the training and validation data sets, by the models that have the higher correlation values.

Model 3 provides the most accurate prediction on the validation data, with an RMS error of 0.00323 (compared to the best validation RMS of 0.00475 for SOGP). Arguably, model 4 provides a better compromise solution. The model has slightly higher training and validation RMS error values than model 3, but has a significantly lower correlation test score. Figure 7-12 shows the results of the five correlation tests for this model (+/- 95% confidence limits are shown as horizontal dashed lines).

Table 7-3 – Preferable individuals obtained using MOGP-FF algorithm

Model no.	RMS	Correlation tests	Validation RMS
1	0.002309	3.3707	0.003507
2	0.002353	3.1662	0.003579
3	0.002388	1.3484	0.003234
4	0.002423	0.7742	0.003450
5	0.002720	0.6392	0.003447
6	0.002825	0.4598	0.003691
7	0.002859	0.2848	0.003645
8	0.002908	0.1603	0.003838
9	0.003434	0.0667	0.004079
10	0.003468	0.0158	0.004172
11	0.003938	0	0.004588

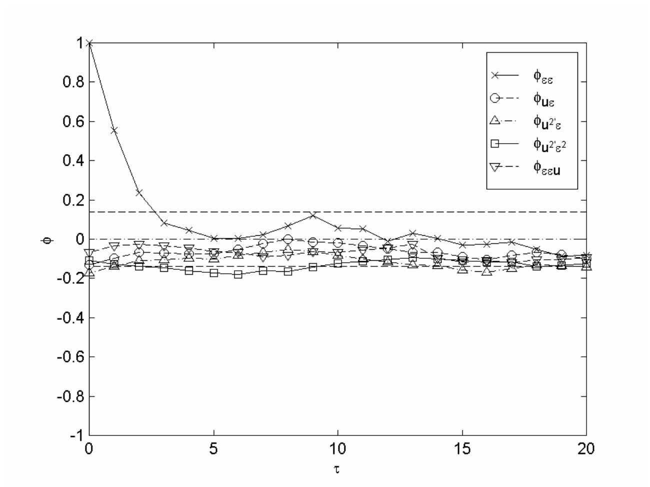


Figure 7-12 – Correlation test plot for model 4 (RMS=0.00242, Φ=0.774)

7.3.1.2 Cooking extruder

Twenty runs of the SOGP and MOGP-P algorithms were performed with a population size of 100 individuals for 100 generations. The non-dominated fronts evolved by the algorithms are compared in Figure 7-13. The plots show that the MOGP-P algorithm was able to produce models with lower correlation values, although some of the prediction errors are rather high.

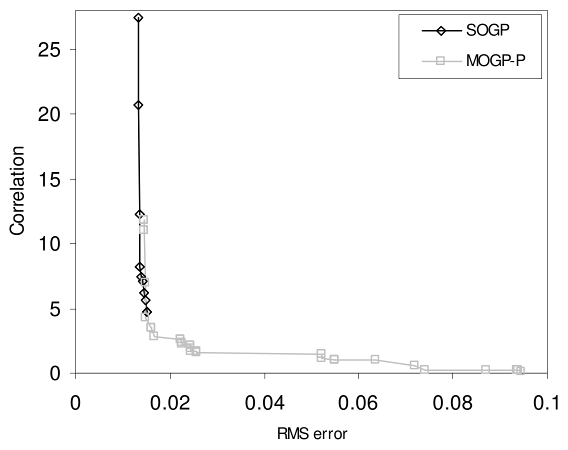
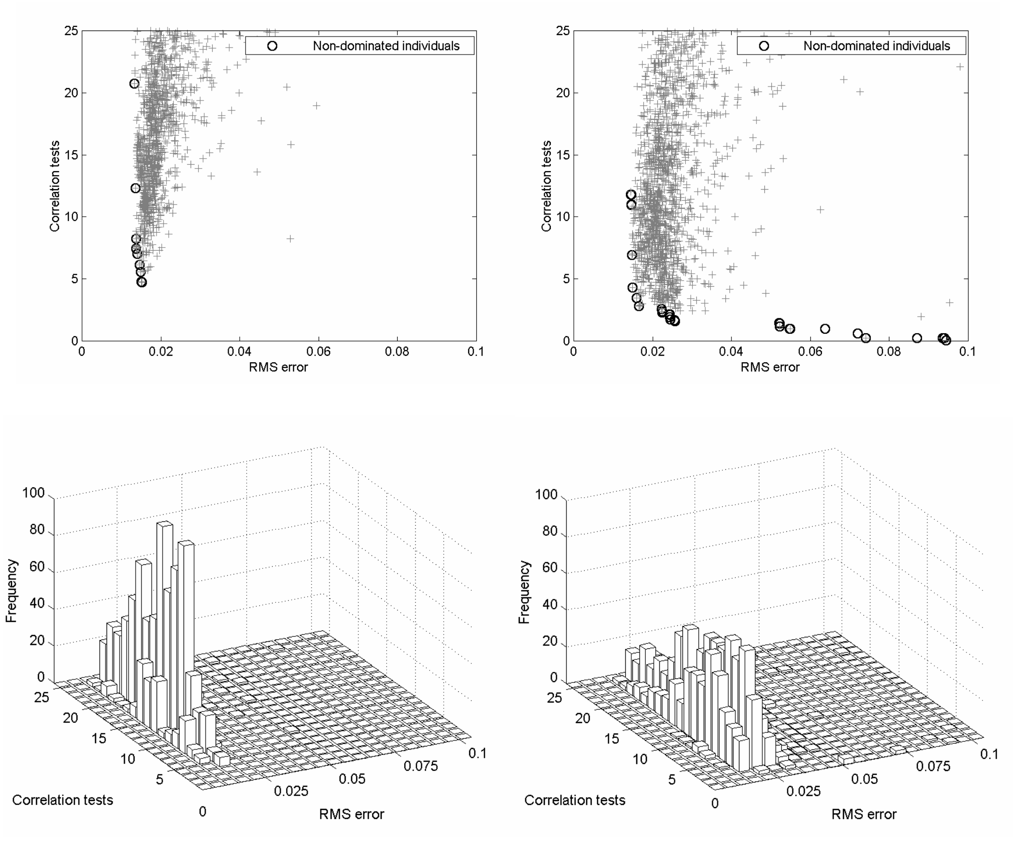


Figure 7-13- Non-dominated fronts achieved by SOGP and MOGP-P algorithms

The distributions of the final generation population members' objective values are shown in Figure 7-14. The distributions reveal that the individuals evolved by the MOGP-P algorithm are all grouped along one half of the non-dominated front. The MOGP-P distribution is wider along the RMS error axis than the SOGP distribution as model accuracy has been reduced at the expense of improved residual test performance. Figure 7-14 also shows that the solutions generated by both algorithms have correlation tests values that are much higher than those achieved in the previous test system example (see Figure 7-9b). This would be expected, as this case study is based on a more complex system. In addition, the correlation objective value is the sum of a larger number of tests as the process has multiple inputs.

In the previous example, the MOGP-P algorithm was able to generate a large number of solutions that performed well in terms of the correlation tests but relatively poorly with respect to RMS error. Consequently, preference information was needed to prevent the algorithm from carrying out a search that was biased towards the residual tests. In this example, preference information is not required for this reason. However, the non-dominated front does contain a number of solutions that have poor RMS values and preference information can be used to ensure that these individuals are discarded and do not form part of the final set of candidate solutions. Another requirement is to improve performance with respect to the residual tests. One possible approach is to set an RMS error goal to force the algorithm to concentrate solely on the correlation tests once a certain level of prediction accuracy has been achieved. Although this means that model accuracy may have to compromised at the expense of the residual tests, it is important to note that the most accurate models on the training data do not necessarily have the best performance on the validation data.



(a) Single objective (b) MOGP-P algorithm

Figure 7-14 – Distributions of final population objective values

An attempt can be made to accomplish both of these aims by setting preference information at two different priority levels. The goal values for each objective and priority are shown in Table 7-4.

When ranking the population, the MOGP algorithm uses the highest priority goals first (priority 2 in this case). At this priority, the correlation objective has been given an infinite value which means that that every population member will satisfy this goal. As a result, the population will be ranked using their RMS values alone. This strategy is intended to prevent the survival of solutions that pass the correlation tests but have higher RMS values.

Table 7-4- MOGP preference information

Priority level	RMS goal	Correlation test goal
2	0.03	∞
1	0.02	0

Ranking will be based solely on RMS values unless comparison is made between two individuals that both satisfy the RMS goal of 0.03. When this occurs, both members satisfy all priority 2 goals and the ranking algorithm moves down to the next priority level. At priority level 1, the correlation goal is zero and is therefore unsatisfied. This means that the correlation test results are now included in the ranking process. Ranking will continue to use both objectives unless two members meet the RMS goal of 0.02, in which case ranking will be performed using correlation values alone. Here, the aim is to force the algorithm to evolve more models that perform well in terms of the correlation tests instead of continuing to evolve models with RMS errors below the 0.02 value. This technique assumes that the models with RMS errors in the range [0.02 0.03] provide an adequate level of prediction accuracy and further reduction in model accuracy can be sacrificed in exchange for improved performance with respect to the residual tests. The choice of RMS goal values is not as arbitrary as may first appear as work carried out in chapter 5 showed that the lowest validation RMS errors were achieved by models that had training errors in this range.

An additional set of twenty runs was performed using the MOGP-FF algorithm using the preference information outline in Table 7-4. Figure 7-15 shows how the evolved non-dominated front compares with the results obtained using the SOGP and MOGP-P algorithms.

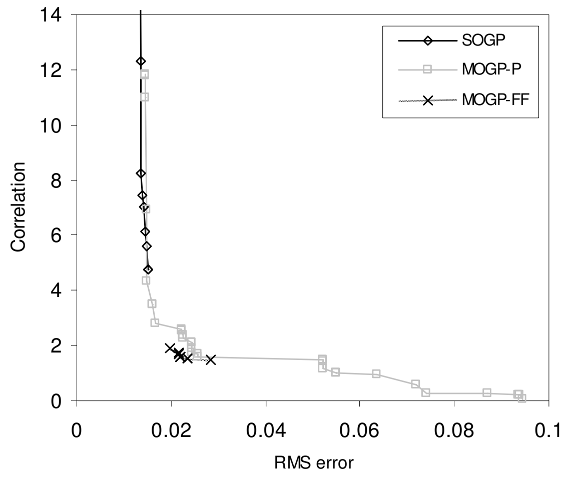


Figure 7-15 – Comparison of non-dominated fronts achieved using MOGP-FF, MOGP-P and SOGP algorithms

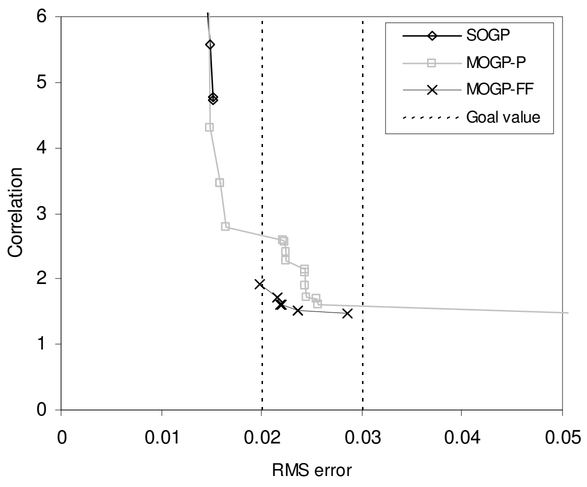


Figure 7-16 – Non-dominated individuals with RMS errors between 0.02 and 0.03

The MOGP-FF non-dominated front covers a much smaller region than those obtained using the SOGP and MOGP-P algorithms. An equally small subset of individuals could be obtained by re-ranking the SOGP and MOGP-P non-dominated individuals using the preference information described in Table 7-4. This process would effectively discard individuals with RMS errors outside the range [0.02 0.03]. Figure 7-16 highlights the non-dominated fronts in this range of RMS values. It can be seen that the MOGP-FF front has lower correlation values than those produced by the MOGP-P algorithm. Although the difference is small, this example still illustrates how the goal based ranking scheme can be used to channel the algorithm's search effort towards the chosen area of the objective space.

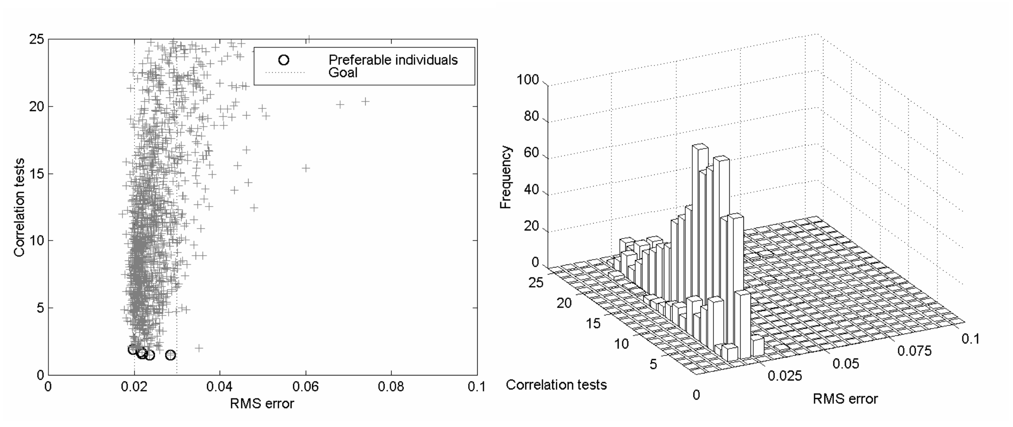
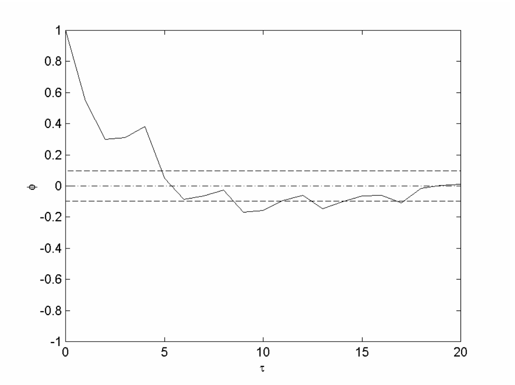


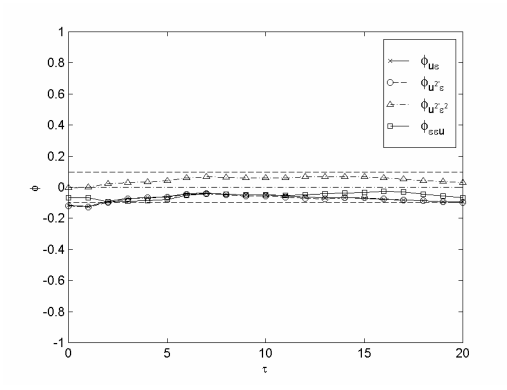
Figure 7-17 – Distribution of final population objective values for MOGP-FF algorithm runs

The distribution of the final population members taken from each MOGP-FF run is shown in Figure 7-17. The inclusion of preference information has resulted in a narrower distribution with the majority of the individuals lying between the two RMS goal lines, with an especially high concentration along the 0.02 line. This is because the search is being targeted towards the correlation test values once this level of prediction accuracy has been achieved. The distribution has also shifted slightly in the direction of lower correlation values.

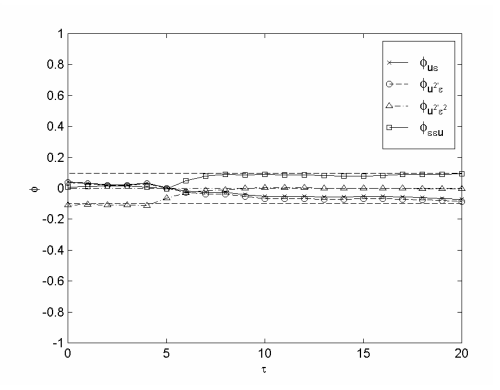
The correlation test plots for the non-dominated MOGP-FF model with the highest correlation objective value are shown in Figure 7-18. The tests that include the four process inputs all have values that fall with in the 95% confidence limits. This indicates that the relevant information present in the input data has been used to develop the model. The correlation objective value is greater than zero because of the first test, which tests for the presence of autocorrelated residuals. This may be due to the fact that the information available in the inputs is not sufficient to completely model the process. The remaining unmodelled dynamics may be the cause of the autocorrelated residuals. Another reason could be that the model contains past outputs of the *model* and not past process outputs. This approach may introduce correlations in the residuals and make it more difficult to completely satisfy all of the tests.



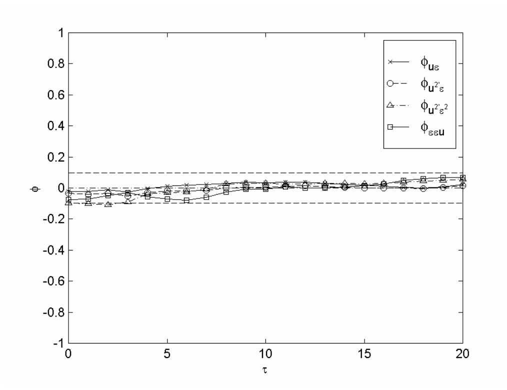
a) Autocorrelation



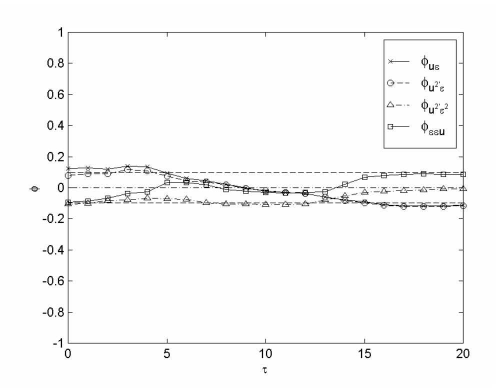
b) Input 1



c) Input 2



d) Input 3



e) Input 4

Figure 7-18 – Example correlation test results (RMS=0.0198, Φ=1.916)

7.3.2 Discussion

A disadvantage of using the correlation tests is that the additional processing time becomes significant as the number of process inputs increases. This is because the four cross correlation tests have to be repeated for each process input. One alternative is to use tests based only on the residuals and the process output. Billings and Zhu (1994) demonstrated how such tests could be used to reduce the number of correlations that have to be evaluated. More recently, Mao and Billings (2000) introduced a new set of model validation tests designed to overcome some of the deficiencies associated with the tests described earlier. It was demonstrated how, under certain conditions, models could pass the original tests, even when the residuals contained predictable information. Consequently, a set of new 'multi-directional' model validity tests was devised in order to overcome this problem. It is possible that these tests could be used within a MOGP framework, the main drawback being an increase in computational effort.

7.4 Conclusions

This chapter has demonstrated, using two test cases, how performance with respect to additional modelling criteria can be improved using a MOGP algorithm. It was found that preliminary algorithm runs using Pareto ranking were useful in order to obtain information regarding the trade-off between the different objectives. This information could then be used to help determine appropriate preference information for the MOGP-FF algorithm. In this respect, the algorithm cannot be considered as an automatic modelling tool as the technique relies heavily on information supplied by the engineer. At this moment in time this may be the best approach as human decision makers have the ability to assess the trade off between the performance measures associated with solutions to complex real-world problems.

Although a Pareto based technique was chosen due to its advantages when compared to methods such as the weighted sum approach, there are still issues that must be addressed to ensure that successful solutions are obtained. The process of setting goal values can be a somewhat arbitrary process, and it is likely that performing additional sets of runs with slightly different values could have yielded further increases in performance. It is possible that an interactive approach with progressive preference articulation would produce a greater number of desirable solutions. However, the process of examining objective information to fine-tune the preference information at the end of every generation would be time consuming. This concern would become more significant for problems that have more objectives and require larger populations sizes.

The work in this thesis has been restricted to case studies consisting of only two-objectives. A number of issues will arise when the algorithm is applied to problems that have a greater number of performance criteria. For example, how does the increased number of performance criteria affect the required population size and number of generations? In addition, it will more difficult to visualise the objective space, making it harder to set goal/priority information effectively. There is a wide range of additional modelling criteria that could be included in the MOGP framework. In terms of residual analysis, there are the additional tests based on the process output and the more recently developed multi-directional tests. Other factors such as the distribution of the residuals could also be considered. In terms of model structure, the maximum lag, degree of non-linearity and the number of model terms have been used by other researchers. Although additional objectives may provide the algorithm with more information about evolved models, a balance must be struck as a large number of objectives may make it difficult to find a suitable compromise solution. Also, objectives based on traditional modelling techniques may not be directly applicable to models generated by a GP algorithm and unexpected results may be achieved.

Another consideration is the extra processing required to evaluate the additional model attributes. Whereas the evaluation of the model string length uses an insignificant amount of processing time, objectives such as correlation tests (which have to be calculated for each input over a number of process lags) carry a greater overhead. A significant amount of the algorithm run time is allocated to the optimisation of the numerical parameters present in each model. One concern is that this process favours the RMS error objective. Results without optimisation and larger population sizes may produce different distributions of individuals and could make it easier to minimise the other objectives.

Another important aspect of this work is that no statistical techniques were used to compare the non-dominated sets of individuals. This means that it is difficult to determine the significance of any differences between two non-dominated fronts. Although there may be difficulties associated with some of the techniques designed to make such comparisons, future work should perhaps investigate the application of statistical MOEA performance measures.

While GP does not provide a significant increase in model accuracy when compared to more established techniques such as neural networks, the algorithm has more of an advantage when applied to multi-objective problems. The parallel nature of GP means that it can evolve a set of candidate solutions with varying levels of performance in each objective. Real world engineering problems typically involve a number of criteria that must be satisfied before a successful solution can be achieved. Consequently, there has been a large increase in the application of MOEAs to engineering problems and it is likely that this trend will continue.

8 Conclusions and Future Work

This thesis has examined a number of aspects of both steady-state and dynamic model development using GP. Chapter 2 discussed how GP could be used to carry out symbolic regression, where models are evolved in the form of tree structured mathematical expressions. Chapter 3 showed how this technique could be applied to the modelling of steady-state chemical processes. The first aim of this thesis was to demonstrate how the modelling capabilities of the standard GP algorithm could be improved. This was achieved using a MBF-GP algorithm, which uses a model structure shared by many existing system identification techniques. Chapter 3 showed that the MBF-GP algorithm was able to generate models of a higher accuracy on the first two case studies, with the standard GP algorithm evolving the best models on the extruder data. The most revealing aspect of the comparison was the difference in the computational cost of each approach. The MBF-GP algorithm required less computational effort to evolve models of the same accuracy as the standard algorithm.

Although the MBF-GP algorithm was generally able to outperform the standard algorithm, it was not able to generate the most accurate models in the extruder case study. It was suggested that the use of a non-linear optimisation routine might give the standard GP algorithm an advantage in some cases. However, the major drawback is that the computational burden is greater than that required by the linear least squares routine used by the MBF-GP algorithm. An important consideration for future algorithm development is the trade-off between the computational complexity and the effectiveness of the available parameter optimisation routines. The MBF-GP algorithm could call on a variety of different techniques, perhaps selected on a probabilistic basis. For example, the L-M algorithm could be used sparingly (not necessarily operating on all of the constants present in a model expression) to provide the ability to fit constants appearing inside basis functions. Simpler operations such as constant mutation could be applied more frequently as they only carry a small processing penalty.

Chapter 4 continued the steady-state modelling work by comparing the performance of the MBF-GP algorithm with neural networks. The results showed that, in terms of prediction error, GP was able to compete in the test system and distillation column case studies, but was outperformed on the extruder data. This shortcoming was addressed by modifying the function set and increasing the population size and number of generations. Although this strategy enabled the algorithm to match the accuracy of the neural network, the additional computational burden was excessively large.

One would perhaps expect GP to require more computational effort than neural networks. Neural network training is essentially a parameter estimation exercise, whereas the GP algorithm performs a search through the space of possible model structures. One of the drawbacks of neural network model development is that a variety of architectures must be examined before arriving at the final model structure. This additional computational effort is not reflected in the FLOPs profiles presented in Chapter 4. If this processing is considered, the difference between the GP and neural network algorithms becomes less significant.

As with any algorithm comparison, it would be beneficial to apply the steady-state modelling algorithms to a wider range of case studies. Future work should include systems that exhibit characteristics typical of those encountered in real industrial applications. For example, data sets collected from a real process may be corrupted with measurement noise and outliers. Another problem encountered in industrial processes is the sheer number of available process variables. It would be interesting to study the performance of GP in response to a large number of inputs. In such a case, the major problem is often one of determining the relevant input variables from a highly correlated data set. In this event, the study may have to be extended to encompass hybrid techniques. For example, a combination of GP and PLS may be better suited to solving this type of problem.

The second aim of this thesis was to demonstrate how GP could be used to evolve models of dynamic processes. Chapter 5 described how the steady-state modelling algorithm could be modified in order to achieve this aim. The most innovative feature of the dynamic version of the algorithm is its ability to automatically discover the appropriate process lags required to build an accurate model. Although discrete-time models have been developed in previous GP applications, the time-shifted model inputs had to be specified explicitly in the terminal set. The aim of this thesis was to develop a more flexible approach so that the final model form is not restricted by assumptions made before the algorithm run. The second case study demonstrated how the GP algorithm was able to combine a number of back-shift operators in order to model the system time delay. This is a significant result, as neural network modelling techniques do not have the ability to perform this task automatically. In practice, this limitation can be overcome by using some form of correlation analysis to identify and remove the time delay before network training. Nevertheless, it is encouraging to discover that GP was able match the accuracy of the neural network that had the benefit of time delay compensation. The GP algorithm also produced some models that gave poor predictions. This should have been anticipated, as the algorithm has to construct the time delay term by nesting a number of back-shift operators. This means that the algorithm has to perform a more difficult task than the neural network and the probabilistic nature of GP means that it is likely to fail on some occasions.

As was found in the steady-state modelling study, the dynamic MBF-GP algorithm required less computational effort to develop models of the same accuracy as the standard algorithm. This is again the result of being able to use a linear optimisation technique (RLS) instead of the more complex non-linear L-M routine used by the standard GP algorithm. The MBF-GP algorithm was able to compete with neural networks in terms of prediction accuracy but was more expensive in terms of computational effort. While this is an important consideration, it should be remembered that computer technology is rapidly improving and today's PCs are capable of tasks that would have previously required access to high-end workstations. The work carried out in this thesis required a relatively modest amount of processing power when compared to other applications of GP. This can be put into context by comparing the single desktop computer used for this work with the network of a thousand PCs employed by Bennett *et al.* (2000) to design electrical circuits.

One of the concerns when attempting to compare GP with neural networks is that it is difficult to carry out an objective study. A number of approaches could improve the performance of both algorithms. In the case of neural networks, examples include alternative training algorithms, weight initialisation techniques and additional methods for improving generalisation. GP practitioners would argue that enhancements could also be made to the GP algorithm. Changes could be as simple as adjustments to the algorithm control parameters or involve the use of alternative genetic operators and parameter estimation techniques. One of the main conclusions to be drawn from the work conducted in thesis is that the relative accuracy of the two techniques appears to be system dependent.

As with all modelling studies, it would have been beneficial to apply the dynamic GP algorithm to a wider range of systems. Future work should ideally study more demanding problems that require the identification of more substantial and/or variable time delays. As with the steady-state modelling algorithm, the dynamic version should be tested on data that is corrupted with noise and outliers. The algorithm could also be used to generate different forms of time series models. For example, the algorithm could be used to generate one-step ahead predictors that could be used to track time varying processes. Future work could also compare the algorithm with other techniques, such as those that use GP to evolve transfer function models.

Despite the encouraging performance of the GP algorithm, the results do not provide sufficient evidence to suggest that GP will become as widely used as neural network modelling techniques. This is largely because neural networks are well established, meaning that the necessary software tools are well developed and widely available. In addition, neural networks have been applied to a wide range of industrial applications and have been the subject of detailed theoretical studies. Although the number of applications of GP is increasing, more work is required in order to study the suitability of GP derived models for real applications. For example, it has been shown that neural networks can be incorporated into a variety of model-based control schemes. It would be interesting to investigate the possibility of using GP models in a similar fashion.

Some researchers have adopted the GP approach to modelling as models are generated in the form of mathematical expressions. Although this model form may be more ‘transparent’ than a neural network, this study has shown how GP can generate models that are rather cumbersome and do not appear to offer any additional process insight. The generic structure of neural network models can be seen as an advantage as the engineer knows that the final model will possess certain properties. For example, an interesting characteristic of RBF networks is that they do not extrapolate into regions outside the range of the training data. In addition, hardware and software implementation of neural network models is aided by the repetitive structure of the network. GP generated models are more unpredictable in the sense that a different model structure is generated by each algorithm run.

One of the key features of the GP algorithm is its ability to perform a highly parallel, population-based search, making it well suited to multi-objective problem solving. The algorithm is able to maintain and evolve a set of candidate solutions that offer different levels of performance with respect to each objective domain. Chapter 7 showed how a MOGP algorithm could account for additional modelling criteria that would usually be considered at the end of the algorithm run. Two case studies were used to make comparisons between the MOGP and SOGP algorithms. The first case study demonstrated how model parsimony could be improved by using the MOGP algorithm. The second case study showed how more complex performance criteria can be included. This was demonstrated using model residual tests, where a number of correlation tests were combined to form a single objective function.

Although Pareto ranking enabled the MOGP algorithm to improve performance with respect to the additional performance criteria, model accuracy was also compromised. This failing was overcome by using a goal-based ranking scheme to guide the algorithm towards the desired region of the search space. This technique is not as straightforward as it first seems, as the need for goal values cannot be envisaged until preliminary runs have been undertaken. For example, initial runs with conventional Pareto ranking revealed that the model string length tended to be minimised at the expense of the prediction error. While this occurrence could easily be explained with hindsight, it is an indication that unexpected results may be achieved when greater numbers of objectives are considered. It may be the case that progressive preference articulation is required, as goal and priority information can be adapted during evolution. The overriding concern with this approach is that it would be time consuming and tedious to carry out multiple algorithm runs. It is also interesting to note how this approach conflicts with the original notion of using GP as a completely automated modelling tool.

Future work with the MOGP algorithm should move on to problems with more than two performance criteria. It would be advisable to begin by including the string length and correlation tests together with RMS error to create a three dimensional problem. This would be a logical starting point as the bi-objective problems concerning these objectives have already been studied in detail. One consideration is the extra processing required to evaluate additional performance measures. Larger population sizes may also be required to compensate for the increased difficulty of the problem. Visualisation of the non-dominated set of individuals will be more difficult, making it harder to configure goal and priority information. It is also important to try to adopt more rigorous techniques designed to compare non-dominated sets of solutions. Again, the difficulties associated with these approaches may be amplified if more than two objectives are being considered. Possible modelling criteria that could be used within a MOGP framework were discussed in chapters 6 and 7.

Multi-objective evolutionary computation is an expanding field of research. While the number of applications is rapidly increasing, so is the need for more theoretical studies to explain the underlying mechanisms associated with this type of algorithm. Research in the GP community has followed a similar path, with a number of recent contributions on the subject of a schema theory for GP (see for example, Poli, 2000). A similar course will have to be taken by researchers in the field of multi-objective problem solving. Although there is empirical evidence to suggest that certain techniques may outperform others, the work tends to concentrate on test cases that do not necessarily imitate the difficulties encountered when tackling complex problems. Also, the metrics used to compare results may not necessarily measure features of algorithm performance that are relevant to real problems.

Recent work has attempted to carry out a more theoretical analysis of MOEA performance. For example, Deb (1999) tackles the problem of comparing MOGAs and describes how test problems can be designed in order to assess certain aspects of algorithm performance. More recently, Laummans *et al*. (2000) provided a model for the role of elitism in MOEAs. There is still a need to formulate methods to analyse different Pareto ranking schemes, fitness sharing and niching methods to provide guidance to engineers wishing to tackle multi-objective problems in this way. Although much of the current work is MOGA based, it is likely that multi-objective

GP will also have an important role to play in solving engineering problems such as process optimisation, control, identification and scheduling which often have multiple design considerations.

A feature of GP that attracts many researchers is that the algorithm can be applied to a diverse range of problem domains with little or no modification to the basic algorithm. While an algorithm that can be used to automatically generate solutions to problems is an attractive prospect, it also encourages the tendency to replace scientific judgement with ever increasing amounts of processing power. This thesis has attempted to demonstrate a more measured approach to problem solving. This is especially significant in the case of multi-objective problems, as a human decision maker is able to influence the evolutionary process. This is an important consideration for future work, which should not rely on the hope that an excessive amount of computing power can be a substitute for engineering insight.

Bibliography

Altenberg, L. (1994) 'Evolving better representations through selective genome growth'. *Proceedings of the First IEEE Conference on Evolutionary Computation*, Orlando, Florida. June 27-29. pp 182-187.

Androulakis, I. P., Venkatasubramanian, V. (1991) 'A Genetic Algorithm Framework for Process Design and Optimisation'. *Computers and Chemical Engineering*, 15, pp 217-228.

Angeline, P. (1997) 'Subtree Crossover: Building Block Engine or Macromutation?'. In *Genetic Programming 1997: Proceedings of the Second Annual Conference (GP97*). J. Koza *et al.* (eds.). San Francisco, Morgan Kaufmann. pp 240–248.

Arkov, V., Evans, C., Fleming, P. J., Hill, D. C., Norton, J. P., Pratt, I., Rees, D., Rodríguez-Vázquez, K. (2000) 'System Identification Strategies Applied to Aircraft Gas Turbine Engines'**.** *Annual Reviews in Control*, 24 (1), pp 67-81.

Babovic, V. (1998) 'A data mining approach to time series modelling and forecasting'. In Babovic and Larsen, (eds.), *Proceeding of the Third International Conference on Hydroinformatics,* pp 847-856, Copenhagen, Denmark, 1998.

Baker, J. E. (1985) 'Reducing Bias and Inefficiency in the Selection Algorithm'. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications.*

Bennett, F. H., Koza, J. R., Yu, J., Mydlowec, W. (2000). 'Automatic synthesis, placement, and routing of an amplifier circuit by means of genetic programming'. *Evolvable Systems: From Biology to Hardware. Proceedings of the Third International Conference, ICES 2000,* Edinburgh, Scotland.

Bettenhausen, K. D., Marenbach, P. Freyer, S., Rettenmaier, H., Nieken, U. (1995) 'Self-Organizing Structured Modelling of a Biotechnological Fed-Batch Fermentation by Means of Genetic Programming'. *IEE Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications,* 12-14 September, 1995. pp 481-486.

Bhat, N., McAvoy, T.J. (1989) 'Use of Neural Nets for Dynamic Modelling and Control of Chemical Process Systems'. *Computers in Chemical Engineering*, 14, pp 573-583.

Bierman, G.J. (1977) *Factorisation Methods for Discrete Sequential Estimation.* Academic Press, New York.

Billings, S.A., Voon, W. S. F. (1986a) 'A Prediction-error and Stepwise-regression Estimation Algorithm for Non-linear systems'. *International Journal of Control,* 44 (3), pp 803-822.

Billings, S. A., Voon, W. S. F. (1986b) 'Correlation based model validation tests for non-linear models'. *International Journal of Control*, 44, pp 235-244.

Billings, S.A., Zhu, Q.M. (1994) 'Nonlinear Model Validation Using Correlation Tests'. *International Journal of Control*, 60 (6), pp 1107-1120

Blickle, T., Thiele, L. (1994) 'Genetic programming and redundancy'. In J. Hopf (ed.), *Genetic Algorithms within the Framework of Evolutionary Computation* (Workshop at KI-94, Saarbrűcken), pp 33-38. Max Planck-Institut fűr Informatik.

Blickle, T., Thiele, L. (1995) 'A Comparison of Selection Schemes used in Genetic Algorithms'. Technical Report 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich.

Blickle, T. (1996) 'Evolving compact solutions in genetic programming: A case study'. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (eds.), *Parallel Problem Solving from Nature IV: Proceedings of the International Conference on Evolutionary Computing,* pp. 564–573, Heidelberg, Germany, Springer-Verlag.

Bremermann, H.J., Anderson, R.W. (1989) 'An Alternative to Back-propagation; A Simple Rule for Synaptic Modification for Neural Net Training and Memory'. Tech. Rep. No.PAM-483, Center for Pure and Applied Mathematics, University of California, Berkeley, CA, USA.

Cao, H., Yu, J., Kang, L., Chen, Yuping, Chen, Yongyan (1999) 'The kinetic evolutionary modelling of complex systems of chemical reactions'**.** *Computers and Chemistry,* 23 (2), pp 143-152.

Charalambous, C. (1992) 'Conjugate Gradient Algorithm for Efficient Training of Artificial Neural Networks'. *IEEE proceedings*, 139 (3), pp 301-310.

Checkoway, C., Kirk, K. (1992) *SIMULINK Users Guide*, The MathWorks, Inc.

Chen, S., Billings, S.A. (1989) 'Representations of Non-linear Systems: the NARMAX Model'. *International Journal of Control*, 49 (3), pp 1013-1032.

Chen, S., Billings, S.A., Cowan, C.F.N., Grant, P.M. (1990a) 'Practical identification of NARMAX models using radial basis functions'. *International Journal of Control*, 52 (6), pp.1327-1350.

Chen, S., Billings, S.A., Grant, P. (1990b) ' Non-linear System Identification Using Neural Networks'. *International Journal of Control*, 51 (6), pp 1191-1214.

Chen, S., Yeh, C. (1997) 'Using genetic programming to model volatility in Financial time series'. In Koza, J., Deb, K., Dorigo, M., Fogel, D., Garzon, M., Iba, H. and Riolo, R. (eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, San Francisco: Morgan Kaufmann, 1997, pp 58-63.

Chipperfield, A. J., Fleming, P. J. (1995) 'Gas Turbine Engine Controller Design Using Multiobjective Genetic Algorithms'. *Proceedings of the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovation and Applications (GALESIA).*

Coello, C. (1999) 'A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization techniques'. *Knowledge and Information Systems,* 1 (3), pp 269-308.

Cramer, N.L. (1985) 'A Representation for the Adaptive Generation of Simple Sequential Programs'. J. J. Grefenstette. (ed.), *Proc. International Conference on Genetic Algorithms and Their Applications*, pp 183-187.

Deb, K. (1999) 'Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems'. *Evolutionary Computation*, 7 (3), pp 205-230.

Doherty, S. K., Gomm, J. B., Williams, D. (1997) 'Experiment design considerations for non-linear system identification using neural networks'**.** *Computers and Chemical Engineering,* 21 (3), pp 327-346.

Draper, N. R. and Smith, H. (1981) *Applied Regression Analysis*. Wiley, New York.

Elsey, J., Riepenhausen, J., McKay, B., Barton G. W., Willis, M. J. (1997) 'Modelling and Control of a Food Extrusion Process'. *Computers and Chemical Engineering*, 21, Suppl, pp S361-S366, *Proceedings of the European Symposium on Computer Aided Process Engineering. - ESCAPE-7*,Trondheim, Norway.

Esparcia-Alcázar, A. I., Sharman, K. (1997) 'Evolving Recurrent Neural Network Architectures by Genetic Programming'. *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 89-94, Morgan Kaufmann.

Evett, M., Fernandez, T. (1998) 'Numeric Mutation Improves the Discovery of Numeric Constants in Genetic Programming'. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, University of Wisconsin, Madison, Wisconsin, USA.

Fahlman, S.E. (1989) ‘Faster-Learning Variations on Back-Propagation: An Empirical Study’, in Touretzky, D., Hinton, G, and Sejnowski, T. (eds.), *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, pp 38-51.

Fogel, I. J., Owens, A. J., Walsh, M. J. (1966) *Artificial Intelligence Through Simulated Evolution*. Wiley.

Fogel, D. B. (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* Piscataway, NJ.

Fonseca, C. M., Fleming, P. J. (1995) ‘An Overview of Evolutionary Algorithms in Multiobjective Optimization’. *Evolutionary Computation*, 3 (1), pp 1-16.

Fonseca, C. M., Fleming, P. J. (1996a) ‘Non-linear System Identification with Multiobjective Genetic Algorithms’. *Proceedings of the 19th World Congress of IFAC,* pp 1887-192, San Francisco, California.

Fonseca, C.M., Fleming, P.J. (1996b) ‘On the Performance Assessment of Multiobjective Stochastic Optimisers’. Internal Report. Department of Automatic Control. Sheffield University, UK.

Fonseca, C. M., Fleming, P. J. (1997). ‘Multiobjective Optimization’. In Back, T., Fogel, D. and Michalewicz, Z. (eds.), *Handbook of Evolutionary Computation*, 1, pp C4.5:1–C4.5:9, Oxford University Press, Oxford, England.

Fonseca, C. M., Fleming, P. J. (1998) ‘Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms - Part I: A Unified Formulation’. *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, 28 (1), pp 26-37.

Frank, P. M., Köppen-Seliger, B. (1997) ‘New Developments Using AI in Fault Diagnosis**’.** *Engineering Applications of Artificial Intelligence,* 10 (1), pp 3-14.

Fröhlich, J., Hafner, C. (1996) ‘Extended and Generalized Genetic Programming for Function Analysis’. <http://www.ifh.ee.ethz.ch/~hafner/ggp/ggppaper.zip>.

Fujiki, C., Dickinson, J. (1987) ‘Using the Genetic Algorithm to Generate LISP Source Code to Solve the Prisoners Dilemma’. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, Erlbaum.

Gani, R. C., Ruiz, A., Cameron, I. T. (1986) ‘A Generalised Model for Distillation Columns - I’. *Computers and Chemical Engineering*, 10 (3), pp 181-198.

Garg, S., Gupta, S. K. (1999) ‘Multiobjective optimization of a free radical bulk polymerization reactor using genetic algorithm’. *Macromolecular Theory and Simulations*, 8 (1), pp 46-53.

Gibbons, J.D. (1985) *Nonparametric Methods for Quantitative Analysis*. American Series in Mathematical and Management Sciences, American Sciences Press.

Goldberg, D. E., Richardson, J. (1987), ‘Genetic Algorithms for Multimodal Function Optimization’. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms,* pp 41-49.

Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, Massachusetts.

Goldberg, D.E., Korb, B., Deb, K. (1989) ‘Messy Genetic Algorithms: Motivation, Analysis and First results’. *Complex Systems* 3 (5) pp 493-530.

Goldberg, D. E. (1990) ‘Real-coded genetic algorithms, virtual alphabets, and blocking’. Illinois Genetic Algorithms Laboratory, Dept. of General Engineering, University of Illinois, Urbana, IL, IlliGAL Report 90001.

Gray, G. J., Murray-Smith, D. J., Li, Y., Sharman, K. C., Weinbrenner, T. (1998) 'Nonlinear model structure identification using genetic programming'. *Control Engineering Practice*, 6 (11), pp 1341-1352.

Greeff, D. J., Aldrich, C. (1998) 'Empirical modelling of chemical process systems with evolutionary programming'. *Computers & Chemical Engineering,* 22 (7-8), pp 995-1005.

Hagan, M. T., Menhaj, M. (1994) 'Training Feedforward Networks with the Marquardt Algorithm'. *IEEE Transactions on Neural Networks,*. 5 (6), pp 989-993.

Hajela, P., Lin, C.-Y. (1992) 'Genetic Search Strategies in Multi-criterion Optimal Design'. *Structural Optimisation*, 4, pp 99-107.

Henson, M. A., Seborg, D. E. (1997) *Nonlinear process control*. Prentice Hall.

Hernandez, E., Arkun, Y. (1993) 'Control of Nonlinear Systems Using Polynomial ARMA Models'. *AIChE Journal*, 39, pp 446-460.

Hiden, H.G. (1998) 'Data-based Modelling Using Genetic Programming'. PhD thesis, University of Newcastle-upon-Tyne, UK.

Hinchliffe, M., Hiden, H., Willis, M., McKay, B., Barton, G.W. (1996) 'Chemical Process Systems Modelling Using a Multi-gene Genetic Programming Algorithm'. *Late Breaking Papers, GP '96*, Stanford, USA.

Hinchliffe, M., Tham, M., Willis, M. (1998) 'Chemical Process Systems Modelling Using Multi-Objective Genetic Programming'. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, University of Wisconsin, Madison, Wisconsin, USA.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*. The University of Michigan Press.

Horn, J., Nafpliotis, N., Goldberg, D. E. (1994) ‘A Niched Pareto Genetic Algorithm for Multiobjective Optimization’. In Michalewicz, Z. (ed.), *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp 82–87, IEEE Press, Piscataway, New Jersey.

Howard, E. and Oakley, N. (1995) ‘Genetic programming as a means of assessing and reflecting chaos’. Tech. Report FS-95-01*,* AAAI Press, pp 68-72.

Hwang, C.-L., Masud, A. S. M. (1979) *Multiple Objective Decision Making – Methods and Applications*. Springer Verlag, Berlin, Germany.

Iba, H., Sato, T., de Garis, H. (1994) ‘System Identification Approach to Genetic Programming’. *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*. pp 401–406.

Ishibuchi, H., Murata, T. (1996) ‘Multi-objective Genetic Local Search Algorithm’. *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC ’96)*, Piscataway, NJ, pp 119-124.

Jacobs, R.A. (1988) ‘Increased Rates of Convergence Through Local Learning Rate Adaption’. Neural Networks, 1, pp 295-307.

Jaske, H. (1996) ‘One-step-ahead prediction of sunspots with genetic programming’. In Jarmo T. Alander (ed.), *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)*, 19-23 August, University of Vaasa, Finland. pp 79-88.

Jiang, M., Wright, A. H. (1992) ‘A Hierarchical Genetic System for Symbolic Function Identification’. In *Proceedings of the 24th Symposium on the Interface: Computing Science and Statistics, College Station, Texas.*

Johansson, R. (1993) *System modeling and identification*, Prentice Hall.

Kaboudan, M. A. (1999) 'A Measure of Time Series' Predictability Using Genetic Programming Applied to Stock Returns'. *Journal of Forecasting*, 18, pp 345-357.

Kanjilal, P. P. (1995) *Adaptive Prediction and Predictive Control*. Peter Peregrinus Ltd.

Koza, J. R. (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press

Koza, J. R. (1994) *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press.

Koza, J. R., Bennett, F. H., Andre, D., Keane, M.A. (1999) *Genetic Programming III; Darwinian Invention and Problem Solving*. Morgan Kaufmann.

Krogh , A., Hertz, J. A. (1992) 'A simple weight decay can improve generalization'. *Advances in Neural Information Processing Systems*, 4, pp 950-957, Morgan Kaufmann.

Kulkarni, B. D., Tambe, S. S., Dahule, R. K., Yadavalli, V. K. (1999) 'Consider Genetic Programming for Process Identification'. *Hydrocarbon Processing* 78 (7) pp 89-97.

Kulshreshtha, M. K. (1991) 'Modelling and control of a twin screw food extruder'. Ph.D. Thesis, Dept. of Food Science and Technology, University of Reading, UK.

Laumanns, M., Zitzler, E., Thiele, L. (2000) 'A Unified Model for Multi-Objective Evolutionary Algorithms with Elitism', *Proceedings of the 2000 Congress on Evolutionary Computation,* 1, pp 46-53, Piscataway, NJ.

Lee, D.-G., Kim H.-G., Baek, W.-P., Heung, Soon, H. C., (1997) 'Critical heat flux prediction using genetic programming for water flow in vertical round tubes'. *International Communications in Heat and Mass Transfer,* 24 (7), pp 919-929.

Lennox, B. (1996) 'Application of Neural Networks to the Process Industries', PhD thesis, University of Newcastle upon Tyne, UK.

Ljung, L. (1999), *System Identification: Theory for the User*, 2nd edition, Prentice Hall.

Löhl, T., Schulz, C., Engell, S. (1998) 'Sequencing of batch operations for a highly coupled production process: genetic algorithms versus mathematical programming'. *Computers and Chemical Engineering*, 22 (Suppl.), pp S579–S585.

Mao, K. Z., Billings, S. A. (1997) 'Algorithms for minimal model structure detection in nonlinear dynamic system identification'. *International Journal of Control*, 68 (2), pp 311-330.

Mao, K. Z., Billings, S. A. (2000) 'Multi-directional Model Validity Tests for Non-linear System Identification'. *International Journal of Control,* 73 (2), pp 132-143.

McKay, B., Lennox, B., Willis, M., Barton, G., Montague, G. (1996) 'Extruder Modelling: A Comparison of two Paradigms'. *UKACC International Connference on Control '96*, 2, pp. 734-739, IEE, 2-5 September 1996.

McKay, B. (1997) 'Studies in Data-based Modelling'. PhD thesis. University of Sydney, Australia.

McKay, B., Willis, M., Barton, G. (1997) 'Steady-state modelling of chemical process systems using genetic programming'. *Computers and Chemical Engineering*, 21 (9), pp 981-996.

Minai, A. A., Williams, R.D. (1990) 'Back-propagation heuristics: A study of the extended delta-bar-delta algorithm'. *Proceedings of the IEEE Joint International Conference on Neural Networks*, 1, pp 595-600.

Montana, D.J., Czerwinski, S. (1996) 'Evolving Control Laws for a Network of Traffic Signals'. *Genetic Programming, Proceedings of the First International Conference*. July 28-31, 1996, Stanford University, California. pp 333-338.

Narendra, K., Parthasarathy, K. (1990) 'Identification and Control of Dynamic Systems Using Neural Networks'. *IEEE Trans. Neural Networks*, 1, pp 4-27.

Nguyen, D., Widrow, B. (1990) 'Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights'. *Proceedings of the Joint International Conference on Neural Networks*, 3, pp 21-26.

Nordin, P., Banzhaf, W. (1995) 'Complexity compression and evolution'. In L. J. Eshelman (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms,* pp. 310–317. San Francisco, CA, Morgan Kaufmann.

Nordin, P., Francone, F., Banzhaf, W. (1996) 'Explicitly defined introns and destructive crossover in genetic programming'. *Advances in Genetic Programming 2*, pp 111-134, MIT Press, Cambridge, MA, USA.

Nordvik, J. P., Renders, J. M. (1991) 'Genetic Algorithms and their Potential for use in Process Control: A Case Study'. In R. K. Belew and L. B. Booker (eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp 480-486.

Poli, R. (2000) 'Exact Schema Theorem and Effective Fitness for GP with One-point Crossover'. *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann. Las Vegas. pp. 469-476.

Porter, M., Willis, M.J., Hiden, H.G. (1996) 'Computer-Aided Polymer Design using Genetic Programming'. MEng. Research Project, Dept. Chemical and Process Engineering, University of Newcastle upon Tyne, UK.

Press, W.H., Flannery, B.P., Saul, A., Vetterling, W.T. (1992) *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edition, Cambridge University Press.

Raidl, G.R. (1998) ‘A Hybrid GP Approach for Numerically Robust Symbolic Regression’. In *Proceedings of the 1998 Genetic Programming Conference*, Madison, Wisconsin, pp. 323-328.

Rechenberg, I. (1972) *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution,* Stuttgart: Frommann-Holzberg Verlag.

Reed, R. (1993) ‘Pruning Algorithms - A Survey’. *IEEE Transactions on Neural Networks,* 4 (5), pp 740-747.

Riedmiller, M., Braun, H. (1993) ‘A direct adaptive method for faster backpropagation learning: The RPROP algorithm’. *Proceedings of the IEEE International Conference on Neural Network*s.

Rodríguez-Vázquez, K., Fleming, P. J. (1998) ‘Multiobjective genetic programming for gas turbine engine model identification’. *UKACC International Conference on Control ’98,* 1-4 September, IEE, 1988.

Rumelhart, D., Hinton, G., Williams, R. (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1, MIT, Cambridge, Massachusetts.

Sarle, W. S. (1995) ‘Stopped Training and Remedies for Overfitting’. In *Proceedings of the 27th Symposium on Interface*.

Schaffer, J. D. (1985) ‘Multiple Objective Optimization with Vector Evaluated Genetic Algorithms’. In Grefenstette, J. J. (ed.), *Proceedings of the First International Conference on Genetic Algorithms.*

Schaffer, J.D., Caruana, R.A., Eshelman, L.J. (1990) ‘Using genetic search to exploit the emergent behaviour of neural networks’. *Physica D,* 42 (1-3), pp 244-248 1990.

Schwefel, H.-P. (1995) *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology Series. Wiley, New York.

Searson, D., Willis, M., Montague, G. (1998) 'Chemical Process Controller Design Using Genetic Programming'. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp 359-364, Morgan Kaufmann, 22-25 July.

Sharman, K.C., Esparcia Alcazar, A.I., Li, Y. (1995) 'Evolving Signal Processing Algorithms by Genetic Programming'. *IEE Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*. 12-14 September, pp 473-480.

Shaw, K. J., Nortcliffe, A. L., Thompson, M., Love, J., Fonseca, C. M., Fleming, P. J. (1999) 'Assessing the Performance of Multiobjective Genetic Algorithms for Optimization of a Batch Process Scheduling Problem', *Congress on Evolutionary Computation*, pp37–45, IEEE Press, Piscataway, New Jersey.

Smillie, K. W. (1966) *An Introduction to Regression and Correlation*, Academic Press, New York.

Söderström, T, Stoica, P. G. (1989) *System Identification*, Englewood Cliffs, NJ, Prentice-Hall International.

Soule, T., Foster, J. A., Dickinson, J. (1996) 'Code growth in genetic programming'. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. R. Riolo (eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference,* pp 215–223, Cambridge, MA: MIT Press.

South, M. C. (1994) 'The application of genetic algorithms to rule finding in data analysis', PhD thesis, University of Newcastle-upon-Tyne, UK.

Srinivas, N., Deb, K. (1994). 'Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms'. *Evolutionary Computation*, 2 (3), pp 221-248.

Tan, K.C., Murray-Smith, D.J., Sharman, K.C. (1995) 'System Identification and Linearisation using Genetic Algorithms and Simulated Annealing'. *IEE Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*. 12-14 September, 1995. pp 164-169.

Turner, P., Montague, G., Morris, J. (1996) 'Dynamic neural networks in non-linear predictive control (an industrial application)'. *Computers and Chemical Engineering,* 20 (972), pp S937-S942.

Van Veldhuizen, D. A., Lamont, G. B. (2000) 'Multiobjective Evolutionary Algorithms Analyzing the State-of-the-Art', *Evolutionary Computation* 8 (2). pp 125-147

Watson, A. H., Parmee, I. C. (1996) 'Identification of Fluid Systems Using Genetic Programming', *Proceedings of the Second Online Workshop on Evolutionary Computation – WEC2*. pp.45-48.

Watson, J.-P. (1999) 'A Performance Assessment of Modern Niching Methods for Parameter Optimization Problems', GECCO-99: *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, Florida, USA. Morgan Kaufmann. pp. 719-725.

Weigend, A. S., Rumelhart, D. E., Huberman, B. A. (1991) 'Generalization by weight-elimination with application to forecasting'. In: R. P. Lippmann, J. Moody, and D. S. Touretzky (eds.), *Advances in Neural Information Processing Systems* 3, San Mateo, CA: Morgan Kaufmann.

Willis, M.J., Di Massimo, C., Montague, G.A., Tham, M.T., Morris, A.J. (1991) 'Artificial Neural Networks in Process Engineering'. *IEE Proceedings - D*.138 (3).

Willis, M. J., Montague, G. A., Di Massimo, C., Tham, M. T., Morris, A. J. (1992) 'Artificial Neural Networks in Process Estimation and Control'. *Automatica*, 28 (6), pp 1181-1187.

Willis, M., Hiden, H., Hinchliffe, M., McKay, B., Barton, G.W. (1997) 'Systems Modelling Using Genetic Programming'. *Computers and Chemical Engineering*, 21 (Suppl.), pp S1161-S1166, Elsevier Science Ltd.

Wray, J., Green, G. G. R. (1991) ‘How Neural Networks Work: The Mathematics of Networks Used to Solve Standard Engineering Problems’. *Proceedings of the American Control Conference*, 3, pp 1654-1667.

Yam, J. Y. F., Chow, T. W. S. (2000) ‘A Weight Initialisation Method for Improving Training Speed in Feedforward Neural Network’, *Neurocomputing,* 30, pp 219-232.

Zitzler, E., Thiele, L. (1999) ‘Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach’. *IEEE Transactions on Evolutionary Computation*, 3 (4), pp 257–271.

Zitzler, E., Thiele, L., Deb, K. (2000) ‘Comparison of Multiobjective Evolutionary Algorithms: Empirical Results’, *Evolutionary Computation*, 8 (2), pp 173-195.

Appendix

A.1 Levenberg-Marquardt Optimisation

A recognised drawback of gradient descent algorithms is that they can make extremely slow progress as they approach the minimum. One method that can be used to overcome this problem is to perform a search in the direction given by Newton's method. The basic assumption of Newton's method is that when the search is sufficiently close to the minimum, a second order Taylor series expansion can be used to approximate the error surface,

$$\hat{\varepsilon} = \varepsilon(\Theta_i) - \mathbf{j}_i^{\mathrm{T}}(\Theta_i - \Theta) + \frac{1}{2}(\Theta_i - \Theta)^{\mathrm{T}}\mathbf{H}_i(\Theta_i - \Theta) \qquad \text{A-1}$$

Where $\hat{\varepsilon}$ is the estimated error, ε is the actual error, Θ is a vector of parameters (model or network parameters) and Θ_i is a particular set of values for those parameters. **j** is a vector of partial derivatives of the error with respect to each of the parameters (the Jacobian) and **H** is a matrix of second derivatives of the error with respect to the parameters (the Hessian).

Calculation of these derivatives enables the parameters that minimise A-1 (Θ_m) to be found. Hopefully, the same parameters will minimise the real function that the quadratic approximates, thus enabling the algorithm to 'jump' directly to the solution. The additional computational effort required to calculate the second derivatives is avoided by using an approximation to the Hessian, ($\hat{\mathbf{H}}$). The Gauss-Newton search direction is then given by,

$$\begin{aligned} \frac{\partial \hat{\varepsilon}}{\partial \Theta} &= -\mathbf{j}_i + \frac{1}{2}\hat{\mathbf{H}}_i(\Theta_i - \Theta_m) = 0 \\ \Theta_m &= \Theta_i - 2\hat{\mathbf{H}}_i^{-1}\mathbf{j}_i \end{aligned} \qquad \text{A-2}$$

The Levenberg-Marquardt(L-M) algorithm varies smoothly between the extremes of the Gauss-Newton method and steepest descent. Far from the minimum, it is likely that the error surface is not well approximated by a quadratic and steepest decent is used. However, as the algorithm approaches the minimum the Gauss-Newton search direction begins to dominate. The L-M parameter update rule can be represented as,

$$\Theta_{i+1} = \Theta_i - (\hat{\mathbf{H}}_i + \lambda_i \mathbf{I})^{-1} \mathbf{j}_i \qquad \text{A-3}$$

Where a large value of λ gives a step in the steepest gradient direction and a small (approaching zero) λ gives a Gauss-Newton step. A simple heuristic is generally used to adjust the value of λ dynamically during a run. This procedure has been found to work well in practice and L-M has effectively become the standard non-linear least-squares optimisation algorithm (Press *et al.*, 1992).

A.2 Steady-state Process Data

A.2.1 Test System

Plots of the input and output variables for this system are shown below.

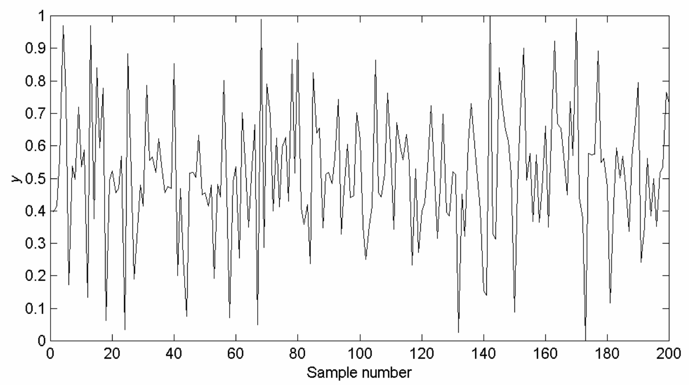


Figure A-1 - Output

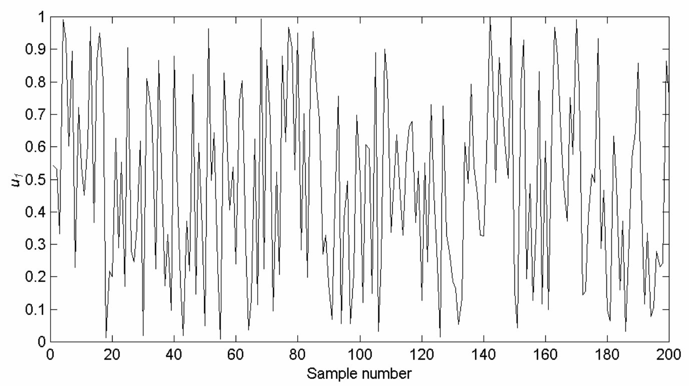


Figure A-2 – Input 1

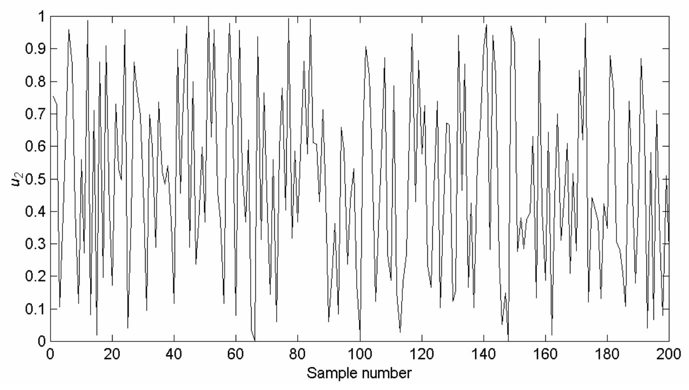


Figure A-3 – Input 2

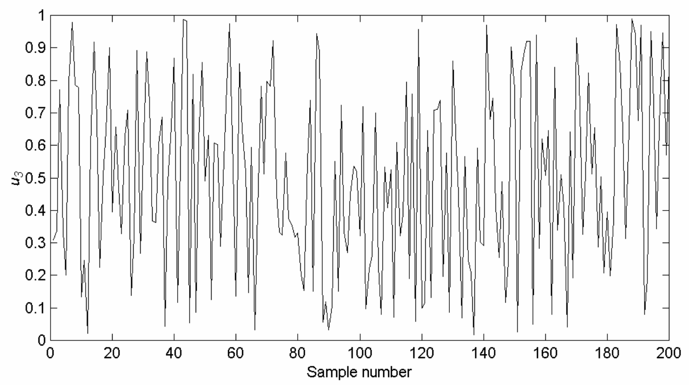


Figure A-4 – Input 3

Table A-1 shows the linear correlations between the input and output variables.

Table A-1– Correlation coefficients for test system

	y	u_1	u_2	u_3
y	1.00	0.60	-0.70	0.03
u_1	0.60	1.00	0.01	0.13
u_2	-0.70	0.01	1.00	0.05
u_3	0.03	0.13	0.05	1.00

A.2.2 Vacuum Distillation Column

Plots of the input and output variables are shown below,

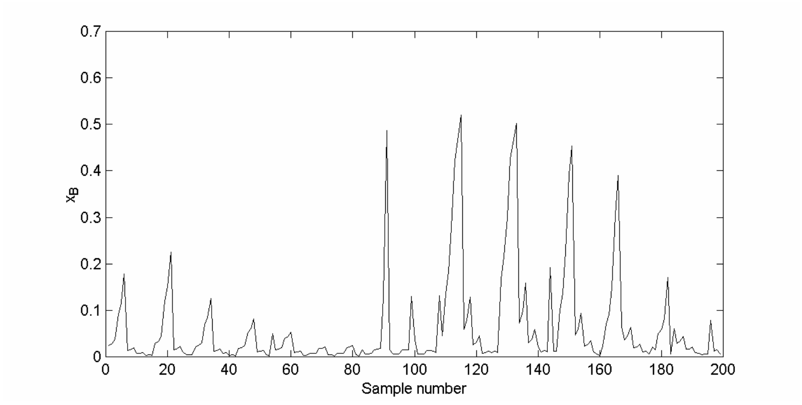


Figure A-5 – Bottom product composition

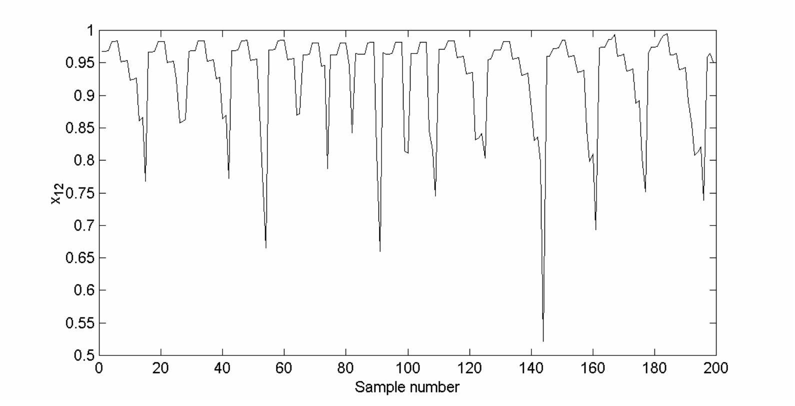


Figure A-6 –composition at tray 12

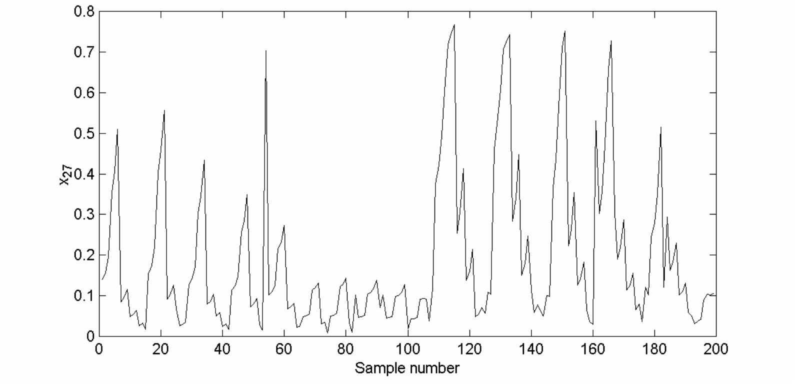


Figure A-7 – composition at tray 27

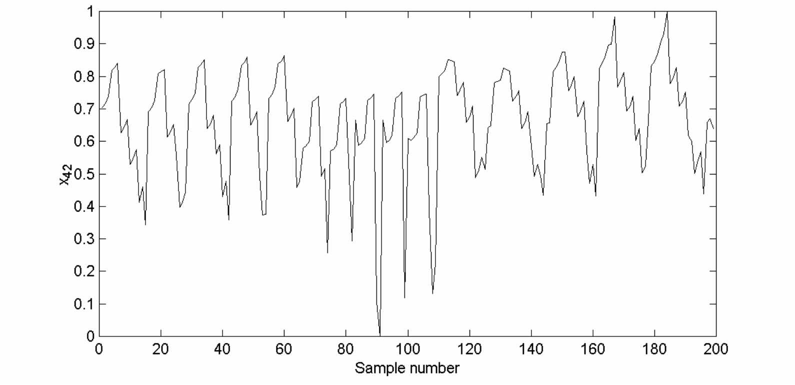


Figure A-8 –composition at tray 42

The linear correlations between the input-and output variables are displayed in Table A-2

Table A-2 – Correlation coefficients for distillation column

	x_B	x_{12}	x_{27}	x_{42}
x_B	1.00	0.12	0.83	0.29
x_{12}	0.12	1.00	0.30	0.79
x_{27}	0.83	0.30	1.00	0.55
x_{42}	0.29	0.79	0.55	1.00

A.2.3 Cooking Extruder

Plots of the input and output variables used in the cooking extruder case study are shown below,

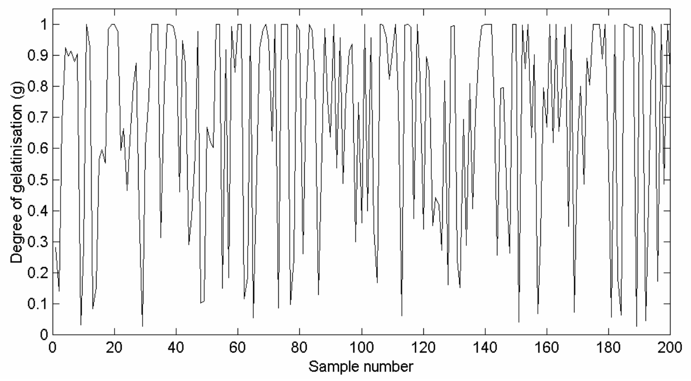


Figure A-9 – Degree of gelatinisation (mass fraction)

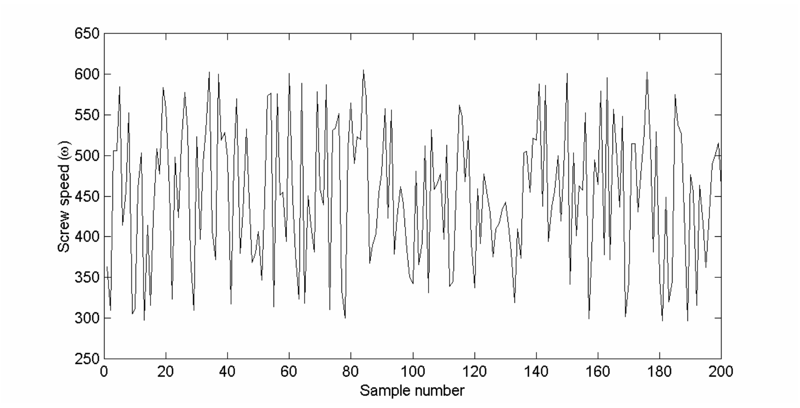


Figure A-10 – Screw speed (rpm)

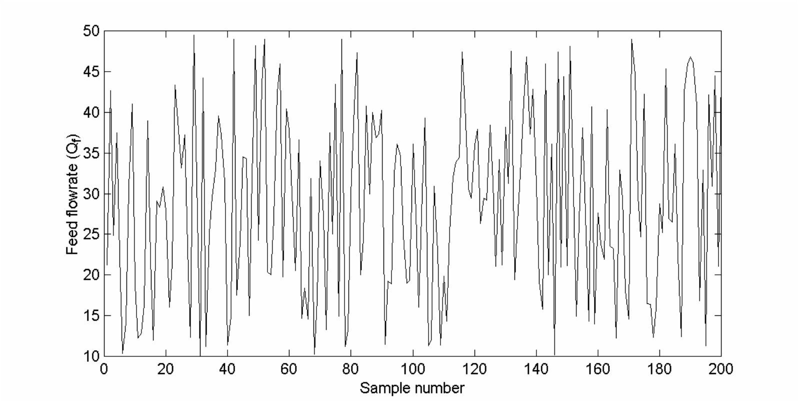


Figure A-11 – Feed flowrate (Kg/s)

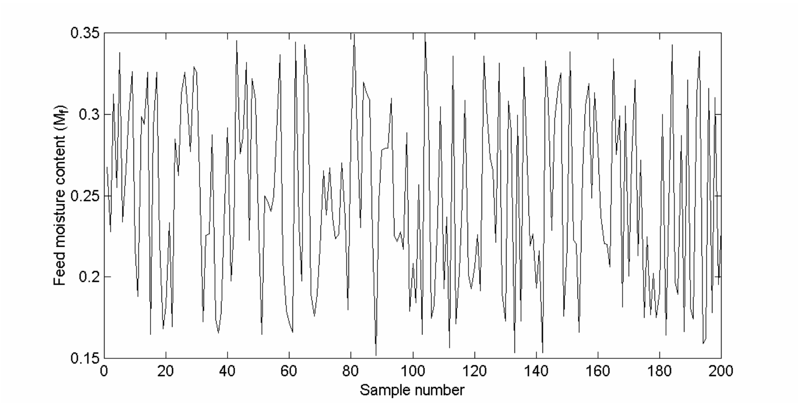


Figure A-12 – Feed moisture content (mass fraction)

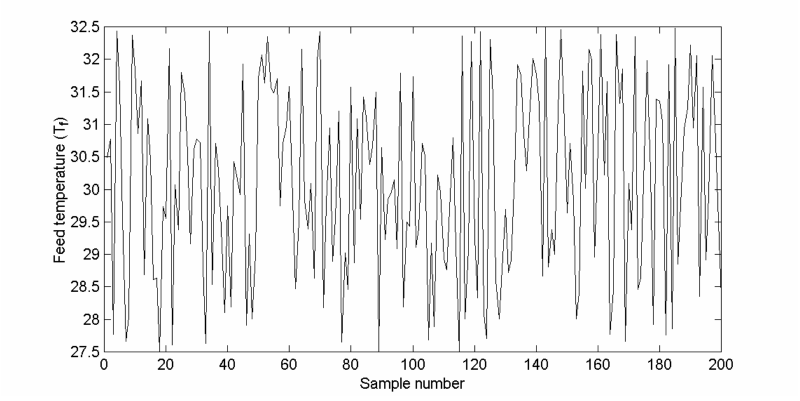


Figure A-13 – Feed temperature (C)

Table A-3 - Correlation coefficients for cooking extruder variables

	g	ω	Q_f	M_f	T_f
g	1.00	0.69	-0.19	-0.60	0.05
ω	0.69	1.00	0.02	0.00	0.14
Q_f	-0.19	0.02	1.00	0.08	0.23
M_f	-0.60	0.00	0.08	1.00	-0.03
T_f	0.05	0.14	0.23	-0.03	1.00

A.3 Linear Models of Steady-state Systems

Linear models were developed for each steady-state data set using batch least squares to determine the regression parameters. RMS values are for the data scaled in the range [0 1].

A.3.1 Test System

A linear model for the test system is shown below. The model has a training RMS of 0.0789 and a validation RMS of 0.0730.

$$\hat{y} = 0.4217u_1 - 0.5122u_2 - 0.0164u_3 + 0.5730 \tag{A-4}$$

The prediction obtained using this model is shown in Figure A-14.

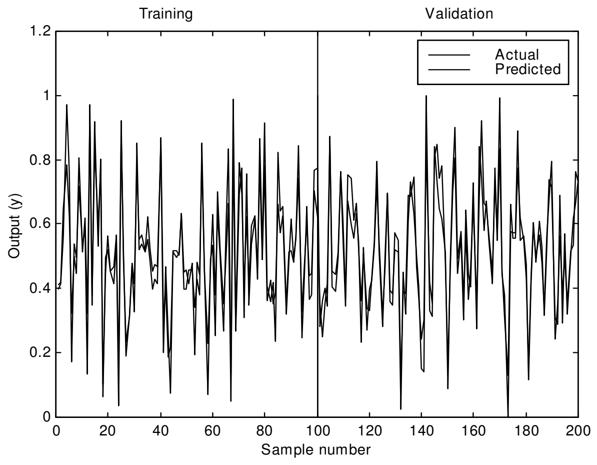


Figure A-14 - Linear model prediction for test system

A.3.2 Distillation Column

A linear model for bottom product composition is shown below,

$$x_B = 0.0217x_{12} + 0.8289x_{27} - 0.3647x_{42} + 0.1458 \qquad \text{A-5}$$

The model has a Training RMS of 0.1093 and a Validation RMS of 0.0890. The model prediction is shown in Figure A-15.

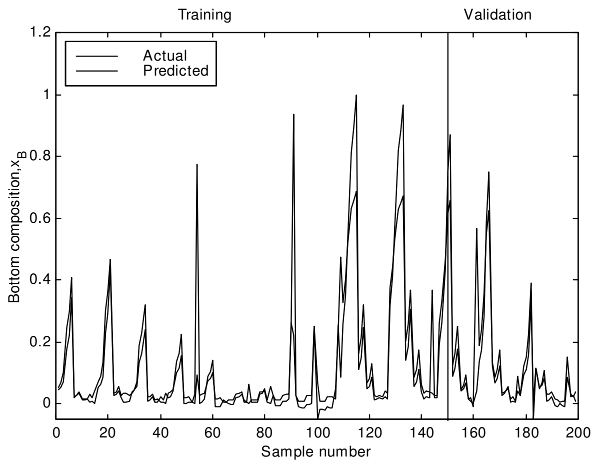


Figure A-15 – Linear model prediction for bottom composition, x_B

A.3.3 Cooking Extruder

Equation A-6 is a linear model for the degree of starch gelatinisation.

$$\hat{g} = 0.8651\omega - 2.036Q_f - 0.5995M_f - 0.0066T_f + 0.6375 \qquad \text{A-6}$$

The model has a Training RMS of 0.1278 and a Validation RMS of 0.1237. The model prediction is shown in Figure A-16.

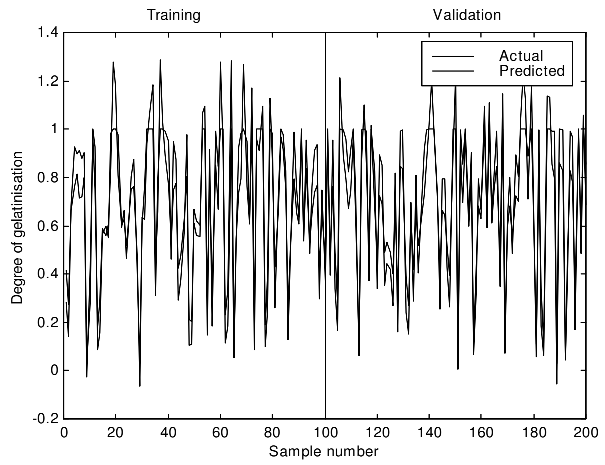


Figure A-16 – Linear model prediction for degree of starch gelatinisation

A.4 Recursive Least Squares Optimisation

This recursive least squares algorithm can be derived from the ordinary least squares estimate as follows (Johansson, 1993),

Consider the regressor ϕ_i and the observations y_i in the following matrices,

$$\Phi_k = \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_k^T \end{bmatrix} \qquad Y_k = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} \tag{A-7}$$

The least squares criterion based on *k* samples is,

$$v(\hat{\theta}_k) = \frac{1}{2}(Y_k - \Phi_k\hat{\theta}_k)^T(Y_k - \Phi_k\hat{\theta}_k) = \frac{1}{2}\varepsilon(\hat{\theta}_k)^T\varepsilon(\hat{\theta}_k) \tag{A-8}$$

The ordinary least squares estimate is given by,

$$\hat{\theta}_k = (\Phi_k^T\Phi_k)^{-1}\Phi_k^T Y_k = (\sum_{i=1}^{k}\phi_i\phi_i^T)^{-1}(\sum_{i=1}^{k}\phi_i y_i) \tag{A-9}$$

Introducing the matrix,

$$P_k = (\sum_{i=1}^{k} \phi_i \phi_i^T)^{-1} = (\Phi_k^T \Phi_k)^{-1} \qquad \text{A-10}$$

A recursive update is given by,

$$P_k^{-1} = P_{k-1}^{-1} + \phi_k \phi_k^T \qquad \text{A-11}$$

$$\begin{aligned} \hat{\theta}_k &= P_k (\sum_{i=1}^{k-1} \phi_i y_i + \phi_k y_k) \\ &= P_k (P_{k-1}^{-1} \hat{\theta}_{k-1} + \phi_k y_k) \\ &= \hat{\theta}_{k-1} + P_k \phi_k (y_k - \phi_k^T \hat{\theta}_{k-1}) \end{aligned} \qquad \text{A-12}$$

It is also possible to calculate the matrix P_k instead of its inverse,

$$P_k = (\Phi_k^T \Phi_k)^{-1} = (\Phi_{k-1}^T \Phi_{k-1} + \phi_k \phi_k^T)^{-1} \qquad \text{A-13}$$

In practice it is often difficult and computationally expensive to perform matrix inversion. However, the problem can be reformulated using the following relationship,

$$(A + BC)^{-1} = A^{-1} - A^{-1} B (I + CA^{-1} B)^{-1} CA^{-1} \qquad \text{A-14}$$

If A-14 is applied to A-13, the following relationship results,

$$\begin{aligned} P_k &= (P_{k-1}^{-1} + \phi_k \phi_k^T)^{-1} \\ &= P_{k-1} - P_{k-1} \phi_k (I + \phi_k^T P_{k-1} \phi_k)^{-1} \phi_k^T P_{k-1} \end{aligned} \qquad \text{A-15}$$

The following three equations are required to carry out RLS optimisation,

$$\hat{\theta}_k = \hat{\theta}_{k-1} + P_k \phi_k \varepsilon_k \qquad \text{A-16}$$

$$\varepsilon_k = y_k - \phi_k^T \hat{\theta}_{k-1} \qquad \text{A-17}$$

$$P_k = P_{k-1} - \frac{P_{k-1}\phi_k \phi_k^T P_{k-1}}{1+\phi_k^T P_{k-1}\phi_k} \qquad \text{A-18}$$

$\hat{\theta}_k$ is the parameter estimate based on the prediction error (ε_k), the regression vector (ϕ_k) and the covariance matrix (P_k).

A.4.1 U-D Factorisation

The matrix P can be expressed as (Kanjilal, 1995),

$$P = UDU^T = \left[UD^{1/2}\right]\left[UD^{1/2}\right]^T \qquad \text{A-19}$$

Where D is a diagonal matrix, U is an upper triangular matrix with 1's on the diagonal and $UD^{1/2}$ is the square root of P. The factorisation is referred to as U-D factorisation of the covariance matrix. Instead of updating P, its factors U and D can be updated and propagated through the recursions. This approach reduces round-off errors and increases numerical stability.

A.5 Dynamic Process Data

This section contains plots of the input-output data used for in the dynamic modelling case studies.

A.5.1 Test system

The input-output data for the test system is shown below. The data has been scaled in the range [0 1].

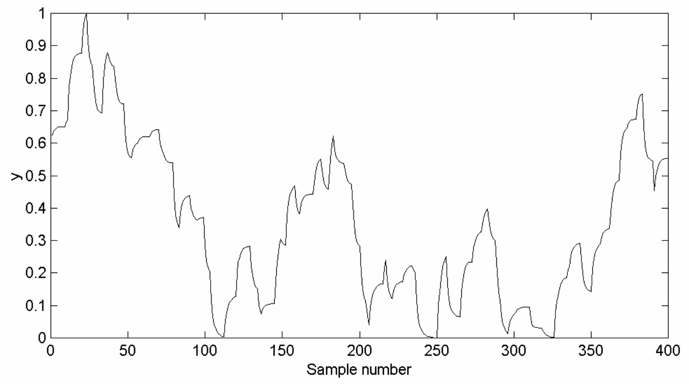


Figure A-17 - Output

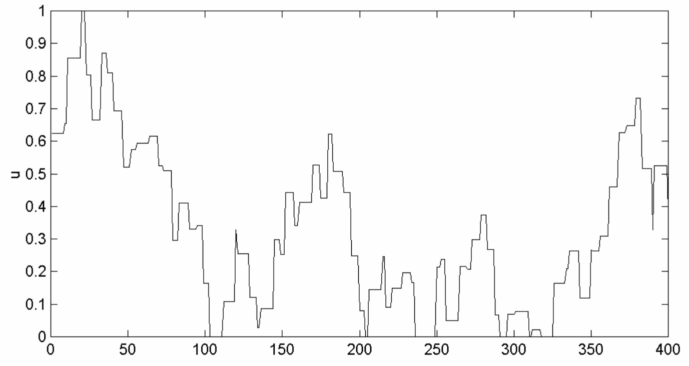


Figure A-18 - Input

A.5.2 Cooking Extruder

The input-output data for the extruder case study is shown in below.

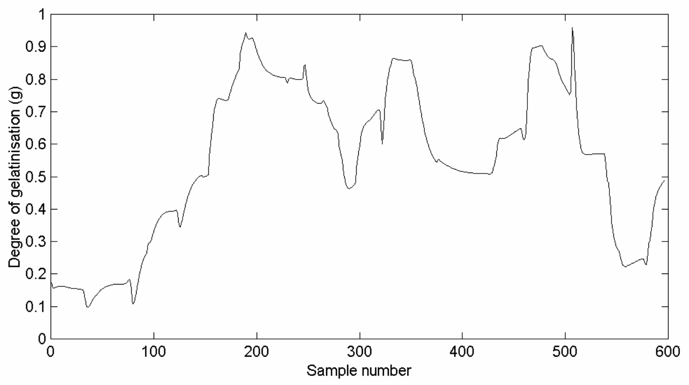


Figure A-19 – Degree of gelatinisation (mass fraction)

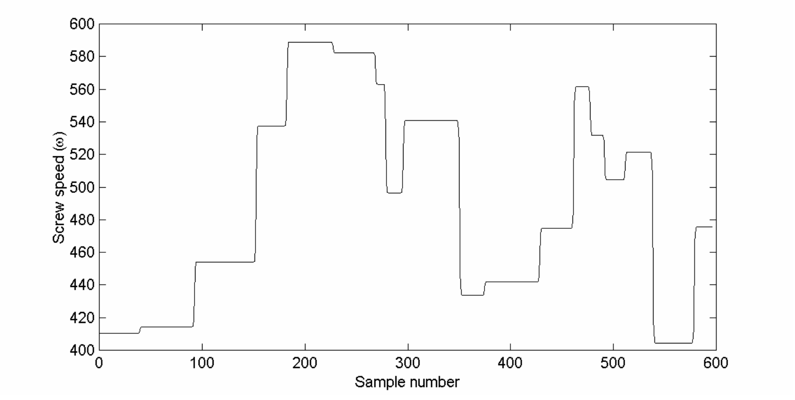


Figure A-20 – Screw speed (rpm)

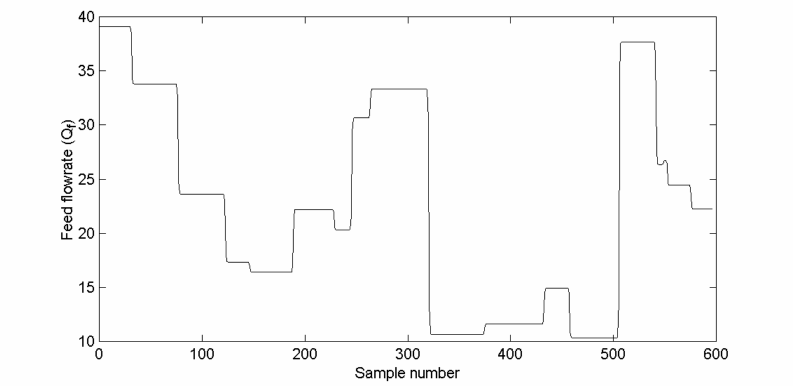


Figure A-21 – Feed flowrate (Kg/s)

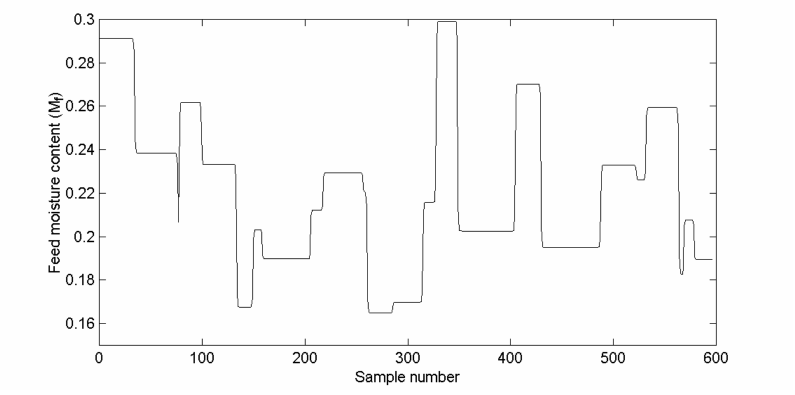


Figure A-22 – Feed moisture content (mass fraction)

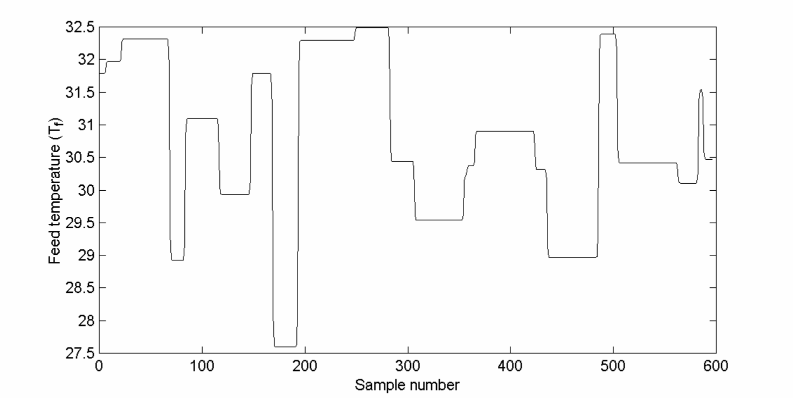


Figure A-23 – Feed temperature (C)

A.6 Linear Models of Dynamic Systems

Linear models were developed using a single time shift for each input/output term. In order to develop models capable of long-term prediction over the entire data set, the model output was used instead of the process output. Model parameters were estimated using the RLS algorithm.

A.6.1 Test System

The following linear model has a training of 0.0173 and a validation RMS of 0.0178,

$$\hat{y}_k = 0.3282\hat{y}_{k-1} + 0.6790u_{k-1} + 0.0140 \qquad \text{A-20}$$

The prediction generated by this model is shown in Figure A-24.

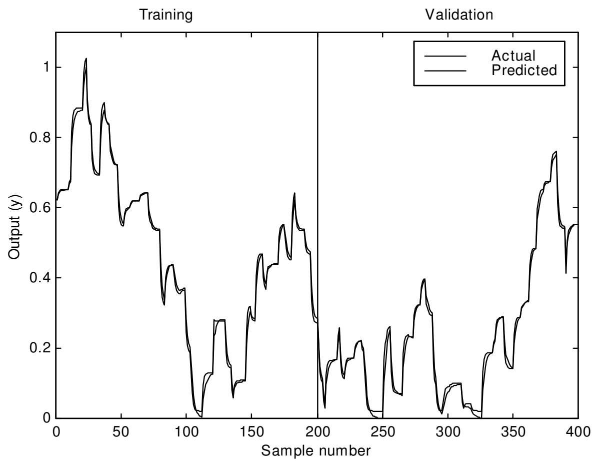


Figure A-24 - Linear model prediction for test system

A.6.2 Cooking Extruder

The following model for the degree of gelatinisation has a training RMS of 0.0659 and a validation RMS of 0.0702

$$\begin{aligned}\hat{g}_k &= 0.8182\hat{g}_{k-1} + 0.1104\omega_{k-1} - 0.0474Q_{f,k-1} \\ &\quad + 0.0036M_{f.k-1} - 0.0434T_{f,k-1} + 0.1010\end{aligned} \qquad \text{A-21}$$

Figure A-25 shows the accuracy of the prediction obtained using this model.

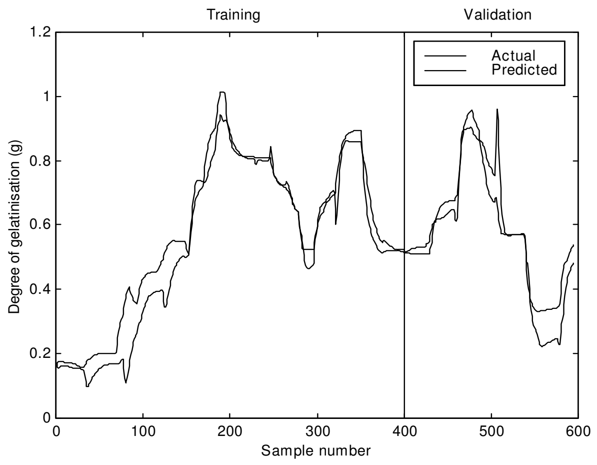


Figure A-25 – Linear model prediction for degree of starch gelatinisation.