View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Warwick Research Archives Portal Repository

THE UNIVERSITY OF WARWICK

University of Warwick institutional repository: http://go.warwick.ac.uk/wrap

A Thesis Submitted for the Degree of PhD at the University of Warwick

http://go.warwick.ac.uk/wrap/49639

This thesis is made available online and is protected by original copyright.

Please scroll down to view the document itself.

Please refer to the repository record for this item for information to help you to cite it. Our policy information is available from the repository home page.

Intelligent Feature Selection for Neural Regression

Techniques and Applications

By

Fu Zhang

Submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

University of Warwick, School of Engineering

May, 2012

For my parents

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisors, Dr. Daciana D. Iliescu and Prof. Evor L. Hines, for their great suggestions and generous support during my research studies and construction of this thesis.

I also wish to thank my parents for their endless support and priceless encouragement.

Declaration

This thesis is presented in accordance with the regulations for the degree of doctor of philosophy. The work described in this thesis is entirely original and my own, except where otherwise indicated.

List of Publications

Zhang, F., Iliescu, D. D., Hines, E. L., Leeson, M. S., & Adams, S. R. (2010). Decision Support System for Greenhouse Tomato Yield Prediction using Artificial Intelligence Techniques. In B. Manos, N. Matsatsinis, K. Paparrizos & J. Papathanasiou (Eds.), *Decision Support Systems in Agriculture, Food and the Environment: Trends, Applications and Advances* (pp. 155-172): IGI Global.

Zhang, F., Iliescu, D. D., Hines, E. L., & Leeson, M. S. (2010). Tomato Plant Health Monitoring: An Electronic Nose Approach. In E. L. Hines & M. S. Leeson (Eds.), *Intelligent Systems for Machine Olfaction: Tools and Methodologies* (pp. 231-248): Medical Information Science Reference.

Zhang, F., Ghaffari, R., Iliescu, D., Hines, E., Leeson, M., & Napier, R. (2011). Field Asymmetric Ion Mobility Spectrometry Based Plant Disease Detection: Intelligent Systems Approach. In R. Qahwaji, R. Green & E. L. Hines (Eds.), *Applied Signal and Image Processing: Multidisciplinary Advancements* (pp. 102-114): IGI Global.

Ghaffari, R., Zhang, F., Iliescu, D., Hines, E., Leeson, M., & Napier, R. (2010). Detection of Diseases and Volatile Discrimination of Plants: An Electronic Nose and Self-Organizing Maps Approach. In E. L. Hines & M. S. Leeson (Eds.), *Intelligent Systems for Machine Olfaction: Tools and Methodologies* (pp. 214-230): Medical Information Science Reference.

Ghaffari, R., Zhang, F., Iliescu, D., Hines, E., Leeson, M., Napier, R., et al. (2010). *Early detection of diseases in tomato crops: An electronic nose and intelligent systems approach*. Paper presented at the The 2010 International Joint Conference on Neural Networks (IJCNN).

Ghaffari, R., Zhang, F., Iliescu, D., Hines, E., Leeson, M., & Napier, R. (2011). The Analysis of Plant's Organic Volatiles Compounds with Electronic Nose and Pattern Recognition Techniques. In R. Qahwaji, R. Green & E. L. Hines (Eds.), *Applied Signal and Image Processing: Multidisciplinary Advancements* (pp. 115-126): IGI Global.

Abstract

Feature Selection (FS) and regression are two important technique categories in Data Mining (DM). In general, DM refers to the analysis of observational datasets to extract useful information and to summarise the data so that it can be more understandable and be used more efficiently in terms of storage and processing. FS is the technique of selecting a subset of features that are relevant to the development of learning models. Regression is the process of modelling and identifying the possible relationships between groups of features (variables). Comparing with the conventional techniques, Intelligent System Techniques (ISTs) are usually favourable due to their flexible capabilities for handling real-life problems and the tolerance to data imprecision, uncertainty, partial truth, etc. This thesis introduces a novel hybrid intelligent technique, namely Sensitive Genetic Neural Optimisation (SGNO), which is capable of reducing the dimensionality of a dataset by identifying the most important group of features. The capability of SGNO is evaluated with four practical applications in three research areas, including plant science, civil engineering and economics.

SGNO is constructed using three key techniques, known as the core modules, including Genetic Algorithm (GA), Neural Network (NN) and Sensitivity Analysis (SA). The GA module controls the progress of the algorithm and employs the NN module as its fitness function. The SA module quantifies the importance of each available variable using the results generated in the GA module. The global sensitivity scores of the variables are used determine the importance of the variables. Variables of higher sensitivity scores are considered to be more

important than the variables with lower sensitivity scores. After determining the variables' importance, the performance of SGNO is evaluated using the NN module that takes various numbers of variables with the highest global sensitivity scores as the inputs. In addition, the symbolic relationship between a group of variables with the highest global sensitivity scores and the model output is discovered using the Multiple-Branch Encoded Genetic Programming (MBE-GP).

A total of four datasets have been used to evaluate the performance of SGNO. These datasets involve the prediction of short-term greenhouse tomato yield, prediction of longitudinal dispersion coefficients in natural rivers, prediction of wave overtopping at coastal structures and the modelling of relationship between the growth of industrial inputs and the growth of the gross industrial output. SGNO was applied to all these datasets to explore its effectiveness of reducing the dimensionality of the datasets. The performance of SGNO is benchmarked with four dimensionality reduction techniques, including Backward Feature Selection (BFS), Forward Feature Selection (FFS), Principal Component Analysis (PCA) and Genetic Neural Mathematical Method (GNMM).

The applications of SGNO on these datasets showed that SGNO is capable of identifying the most important feature groups of in the datasets effectively and the general performance of SGNO is better than those benchmarking techniques. Furthermore, the symbolic relationships discovered using MBE-GP can generate performance competitive to the performance of NN models in terms of regression accuracies.

Abbreviations

AI	Artificial Intelligence
AN	Artificial Neuron
ANN	Artificial Neural Network
BNS	Biological Neural System
BFS	Backward Feature Selection
CI	Computation Intelligence
DM	Data Mining
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EP	Evolutionary Programming
FD	Feature Derivation
FE	Feature Extraction
FFBP	Feed Forward Back Propagation
FFS	Forward Feature Selection
FLS	Fuzzy Logic System
FNN	Forward Neural Network
FS	Feature Selection
FST	Fuzzy System Technique
GA	Genetic Algorithm
GNMM	Genetic Neural Mathematical Method
GP	Genetic Programming
IST	Intelligent System Technique
KD	Knowledge Discovery
MBE	Multi-Branch Encoding
MBE-GP	Multi-Branch Encoding Genetic Programming
MCS	Monte Carlo Simulation
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NN	Neural Network

PC	Principal Component
PCA	Principal Component Analysis
RBF	Radial Basis Function
RBFN	Radial Basis Function Network
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SA	Sensitivity Analysis
SC	Soft Computation
SFS	Sequential Feature Selection
SGNO	Sensitive Genetic Neural Optimisation
SOM	Self Organising Map
TDNN	Time-Delay Neural Network
VPD	Vapour Pressure Deficit
WHRI	Warwick Horticulture Research International

CHAPTER 1

Introduction

1.1 Overview

The modern world is expending at an unprecedented speed from the physical into the virtual. Rapid advances in communication and storage technology make the collection and distribution of data (information) more and more mandatory and convenient. Along with the improvements in data collection, the capacity of data grows larger and the dimensionality of data grows higher. However, our understanding of the corresponding data and the discovery of valuable knowledge hidden in such large amounts of data has been growing at a much lower rate. Without effective data access/extraction and knowledge explanation, this rapid expansion in the amount of data will not become a useful asset (Liu and Motoda, 2008; Pappa and Freitas, 2010).

Data mining (DM) is a relatively new term proposed in recent decades, though the tasks of data mining, such as regression and classification, have existed for a much longer time. Generally speaking, DM refers to the analysis of data to reveal any hidden relationships and construct abstract knowledge, so that the data can be re-expressed in ways that are more understandable and useful to the user (Hand et al., 2001). DM covers a broad range of techniques and is being used to analyse a broad variety of data. Commonly used DM techniques include dimension reduction, regression modelling, model building and evolutionary algorithms

(Larose, 2006). Feature selection (FS) and regression are two important techniques in DM. FS is designed to extract the group of features/variables of the highest relevance from the entire dataset and thus reduce the dimensionality of the original dataset and regression aims to discover any relationship which may exist in the data.

1.2 Tasks and Procedures

There are different ways to categorise DM techniques. For example, Tan, Steinbach and Kumar (2005) divide DM into two categories based on their applications, which are descriptive tasks and predictive tasks; while Hand, Mannila and Smyth (2001) suggested that there should be more groups, such as data exploration, pattern discovery, rule extraction, etc. Wang and Fu (2005) categorise DM techniques into three groups based on their activities, which are dimensionality reduction, classification and clustering, and rule extraction. However, their categories do not cover data regression. This thesis adopts the categories suggested by Wang and Fu (2005) together with regression as a new category.

Dimensionality reduction generally involves FS and Feature Extraction (FE). FE generates new dataset by deriving new features (less than the amount of original ones) to reduce the amount of data and thus increase the computational efficiency. Extensive efforts have been put into FS research in recent years. One of the fundamental motivations of FS is the curse of dimensionality in data. The number of features is the key factor that determines the complexity of the collected data.

The more features, the greater the complexity. In general, a linear increase in the number of features may lead to an exponential increase in the complexity (Pappa and Freitas, 2010). Commonly used dimensionality reduction techniques are Principal Component Analysis (PCA), Genetic Algorithms (GAs) and Sequential Feature Selection (SFS).

Classification and clustering are the techniques of close relationships. The purpose of clustering is to split data into various groups, where the data in the same group are of similarity or close relationship. Data classification takes the process one step further, it builds a model which can then be used to classify unseen data instances (Larose, 2006). Rule extraction is usually connected with classification and clustering and aims to present data in such a way that interpretations are easily understandable and decisions can be made based on the knowledge obtained from the data (Wang and Fu, 2005). Regression is the technique that discovers the relationship between the data fields. In regression, a group of data is used as the regression inputs and the residual data is used as the target. The job of regression is to identify the relationship between the inputs and the target. Figure 1.1 illustrates the general DM procedures.

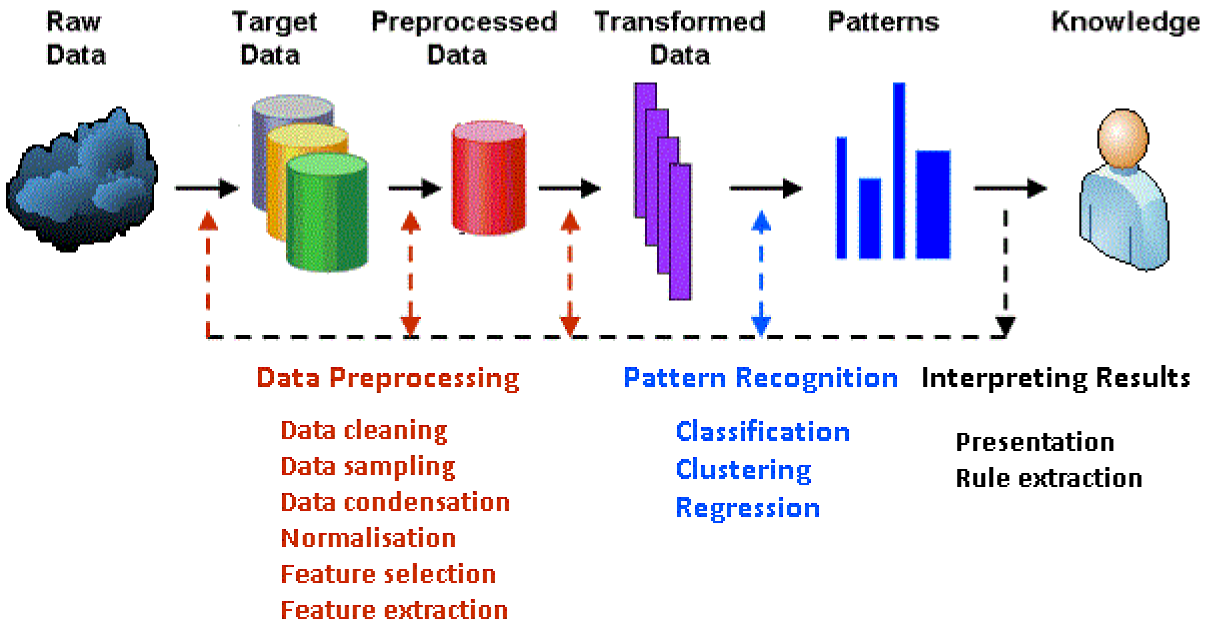

Figure 1.1: General procedures in DM (adapted from Kamath, 2000)

As illustrated in figure 1.1, a complete process that transforms the original raw data into comprehensive presentation or knowledge involves three major steps, which are data preprocessing, pattern recognition and result interpretation. These major steps can be further split into minor processes, which are explained briefly below (Venugopal et al., 2009):

- Data preprocessing is usually the first in data processing, it is generally time consuming, but critical as it determines the quality of the data subset extracted/converted from the original data for further processing. The key techniques in data preprocessing are:
    - Data cleaning. This step aims to remove noise, irrelevant data, and data fields that are not machine understandable, such as text comments typed in along with the numeric measurements, from the collection.

    - Data condensation. This step involves combining datasets under the topic from different sources.
    - Normalisation is the rescaling of data. It brings data fields to a common scale and makes them more comparable.
    - FS and FE may be used interchangeably, but there are minor differences. They are both important dimensionality reduction techniques. FS generally refers to the selection of a group of data fields from the original data without modifying the contents. FE, on the other hand, may refer to not only what FS can perform, but also the extraction (with modification) of the key information from the original data. To sum up, the result of FS is always a subset of the original data, while the result of FE is a new dataset.
    - Data transformation is the step to reorganise the preprocessed data into the forms that are acceptable by the further procedures/techniques.
- Pattern recognition is the most important procedure in DM. It generally concerns building abstract models that can determine patterns from the observed data and the models will be used to induce knowledge. Models are simply computer algorithms; commonly used models include classification, clustering and regression.
- Result interpretation. In this step, comprehensive knowledge is extracted from the proposed models in the pattern recognition step and is presented to the end user in visualised ways or symbolic expressions.

1.3 Intelligent System Techniques (ISTs)

In the modern development of DM techniques, ISTs have drawn the attention of many researchers. More and more IST systems have been developed to assist DM. They have become an important group of techniques in DM (Karray and Silva, 2005; Mitra et al., 2002). Commonly used IST systems in DM include Artificial Neural Networks (ANNs), Evolutionary Algorithm (EAs) and Fuzzy System Techniques (FSTs). These techniques have been successfully applied in DM, especially in the fields of dimensionality reduction, classification, regression and rule extraction (Ruan et al., 2005; Sumathi and Sivanandam, 2006; Venugopal et al., 2009).

The term IST is generally used interchangeably with the terms Soft Computing (SC) and Computational Intelligence (CI). ISTs are usually favourable to solve problems due to their flexible information processing capabilities for handling real-life problems. They differ from the conventional techniques in that they are tolerant to data imprecision, uncertainty, approximate reasoning, and partial truth (Mitra et al., 2002; Venugopal et al., 2009). Fuzzy sets in FST can provide a natural framework in dealing with uncertainties; ANNs are widely used for classification, rule generation and regression. GAs are generally involved in various optimisation and search applications, such as finding the global minimum (Mitra et al., 2002). These techniques, ANNs, GAs and FST, are generally considered as the fundamental components of IST. Many researchers have attempted to develop hybrid ISTs by using these fundamental components cooperatively rather than exclusively and competitively to obtain better efficiency and robustness (Harris et al., 2002; Mitra et al., 2002; Yang, 2010).

1.4 Research Objectives

The overall objective of this thesis is to develop a general purpose IST system which can perform FS effectively to reduce the dimensionality of the original dataset while maintaining flexible amounts of information carried by the original dataset. Along with the discovery of effective features, regression models are constructed to determine the relationship between the selected features and the target data. In addition, symbolic expressions (symbolic rules) between the features and the target data are discovered.

Apart from the theoretical development of the IST system, this thesis also aims to explore the possible applications and limitations of the SGNO and the associated techniques in plant science, civil engineering and economics. In addition, the thesis also conducts benchmark studies between the SGNO and several other FS techniques to test the performance of the SGNO.

The unique contributions of this thesis are summarised below:

- Developing a general purpose hybrid IST method, the Sensitive Genetic Neural Optimisation (SGNO), which ranks the features in a dataset in terms of their importance for creating mathematical models of the dataset.
- Building mathematical models using ANNs to discover the relationships between input features and output features in datasets.

- Discovering and expressing the relationships between input features and output features using mathematical operators, such as plus, minus, times, etc.
- Demonstrating the application of SGNO in various areas, including horticulture, engineering and economics.

1.5 Thesis Outline

The current chapter presents a brief overview of DM concepts, including FS and regression, and the general applications of ISTs in the field of modern DM. In addition, the general research objective and the overall structure of the thesis are also discussed in this chapter.

Chapter 2 briefly reviews some fundamental IST techniques that are relevant to the development of the proposed hybrid technique, including ANNs, GAs and Genetic Programming (GP). In addition, another hybrid system, namely the Genetic Neural Mathematical Method (GNMM) is also reviewed in this chapter and will be used as one of the benchmarking techniques in the application chapters.

Chapter 3 explains the implementation of the proposed SGNO technique in detail and the construction of a GP based symbolic regression system, which aims to extract symbolic rules from the result generated by SGNO. In addition, the implementations of the benchmarking techniques are discussed in this chapter as well.

Chapter 4 demonstrates the application of SGNO in horticulture concerning the short term prediction of the weekly greenhouse tomato yield. The performance of SGNO is benchmarked with several FS techniques.

Chapter 5 illustrates an application of SGNO in engineering to predict the longitudinal dispersion coefficients in natural streams.

Chapter 6 presents another application of SGNO in engineering to model the relationship between the overtopping discharges and several structural and hydraulic properties of coastal structures.

Chapter 7 demonstrates the feasibility of SGNO to produce economic models. In this chapter, SGNO is used to model the relationship between the growth of the industrial inputs and the growth of the gross production output.

Chapter 8 concludes the discoveries presented in the previous chapters and suggests some possible directions of further research.

References

Hand D.J., Mannila H., Smyth P. (2001) Principles of Data Mining The MIT Press.

Harris C.J., Hong X., Gan Q. (2002) Adaptive modelling, estimation, and fusion from data : a neurofuzzy approach Springer.

Kamath C. (2000) Sapphire: Large Scale Data Mining and Pattern Recognition, Lawrence Livermore National Laboratory.

Karray F.O., Silva C.W.D. (2005) Soft Computing and Intelligent Systems Design: Theory, Tools and Applications Addison-Wesley.

Larose D.T. (2006) Data Mining Methods and Models WILEY.

Liu H., Motoda H. (2008) Computational Methods of Feature Selection Chapman & Hall.

Mitra S., Pal S.K., Mitra P. (2002) Data Mining in Soft Computing Framework: A Survey. IEEE Transactions on Neural Networks 13:3-14.

Pappa G.L., Freitas A.A. (2010) Automating the Design of Data Mining Algorithms: An Evolutionary Computation Approach Springer.

Ruan D., Chen G., Kerre E.E., Wets G. (2005) Intelligent data mining: techniques and applications Springer.

Sumathi S., Sivanandam S.N. (2006) Introduction to Data Mining and its Applications Springer.

Tan P.N., Steinbach M., Kumar V. (2005) Introduction to Data Mining. first ed. Addison-Wesley Longman Publishing.

Venugopal K.R., Stinivasa K.G., Patnaik L.M. (2009) Soft Computing for Data Mining Applications Springer-Verlag, New York.

Wang L., Fu X. (2005) Data Mining with Computational Intelligence Springer.

Yang J.H. (2010) Intelligent data mining using artificial neural networks and genetic algorithms: techniques and applications, Engineering, University of Warwick, Coventry. pp. 260.

CHAPTER 2

Intelligent Systems Techniques

2.1 Overview

In the previous chapter, the general concept and procedures of DM were introduced and the research objectives of the thesis were outlined. This chapter provides the theoretical background of some widely used ISTs that are relevant to the development of SGNO.

The term 'intelligence' is difficult to define as different people may have different personal definitions for the term. In the popular sense, superior mental ability to interact with the environment and solve problems is considered as the sign of intelligence. Thus, 'intelligence' is usually defined as the ability to learn from the external environment, evaluate, judge and apply the knowledge/experience to manipulate the environment. The general abilities that are referred to as the behaviours of intelligence include the following abilities (Nilsson, 1998):

- to adapt to a new environment
- to acquire knowledge
- to evaluate and judge
- to think in an abstract manner
- to think productively

The term 'Artificial Intelligence' (AI) is again difficult to define. However, it is understood to be broadly concerned with the intelligent behaviours of humans. The intelligent behaviours include learning, reasoning, communicating and acting in external environments. In the aspect of engineering and computer science, AI generally focuses on the concepts and ideas underlying the design of intelligent machines/systems. This chapter concentrates on several Intelligent System Techniques (ISTs), which are an offshoot of AI. The main paradigms of IST include Artificial Neural Networks (ANNs), Evolutionary Computation (EC), Fuzzy Systems (FS) and Swarm Intelligence (SI) (Engelbrecht, 2007).

The following sections of this chapter provide the basic theoretical background of some IST techniques that are the key components of SGNO, including ANNs and EC. These techniques have been applied successfully in a wide range of applications, including prediction, pattern recognition, feature extraction, optimisation, etc.

2.2 Artificial Neural Networks (ANNs)

The brain is one of the most important and complex organisms for human beings. It is well known that biological neural systems (BNSs) can perform extraordinarily complex tasks and are capable of learning over time (Berthold and Hand, 2007; Haykin, 1999). The implementation of ANNs was inspired from brain modelling studies.

2.2.1 Biological Roots

Neurons are the biological cells that enable the brain to perform all the complex functions. There is of the order of 10-500 billion neurons and 60 trillion synapses in the human cortex performing pattern recognition, perception, body movement control, etc. during daily life. Each of these biological neurons is connected to about 10 thousand others in highly complex manners (Beale and Jackson, 1990; Parks et al., 1998).

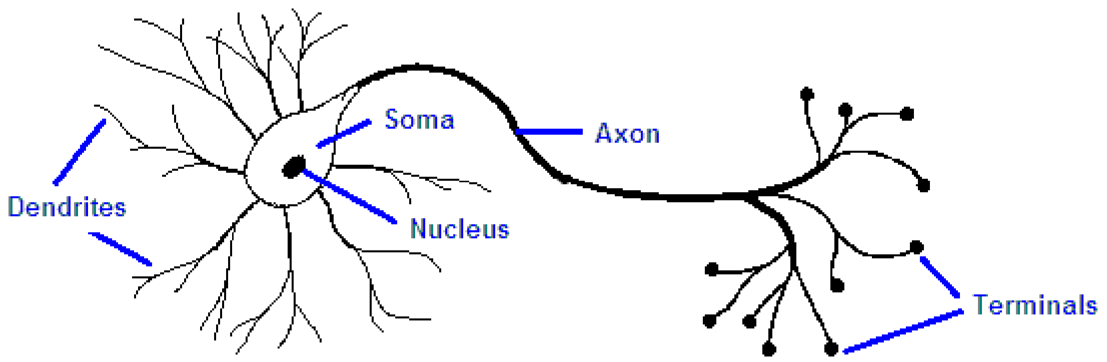

Figure 2.1: Basic representation of a biological neuron

Under the microscope, neurons appear to be of different shapes and sizes. However, all the neurons have similar basic structure and the different regions of the neuron have specific functions. Figure 2.1 illustrates the basic structure of a biological neuron. The dendrites receive biological electrical signals from other neurons. The soma integrates, processes these incoming signals, and then conveys the resulting information to the terminals along the axon. At the terminals, chemical substances, which are known as neurotransmitters, are released to activate the communication with adjacent neurons (Richards et al., 2007).

In order to perform specific functions, such as process information and producing appropriate responses, individual neurons must be connected to each other to form large networks. The connectivity between neurons is referred as a synapse. Learning is thought to be achieved by modifying the strengths of effective couplings at synapses between neurons. The modification of couplings to reinforce good connections is an important feature of ANNs (Beale and Jackson, 1990; Haken, 1996).

2.2.2 Artificial Neurons (ANs)

As ANNs reflect simplified abstract mathematical models of the nervous systems, the smallest part/unit of an ANN is a simple arithmetic processing unit, namely an Artificial Neuron (AN), which is the mathematical model of a biological neuron. Each AN is responsible for acquiring information from one or more input signals and generating a single output signal. There are three basic elements in the neuronal model, which are listed below:

- A set of connection links (inputs). These links represent the connections to other neurons and receive input signals from those neurons. Each of these connection links is characterised by a weight or strength.
- A linear combiner or adder which sums the input signals weighted by their corresponding weights.
- An activation function which limits the amplitude of the output signal by mapping the signal generated by the linear combiner to a signal within a certain range. The activation function could be linear or non-linear.

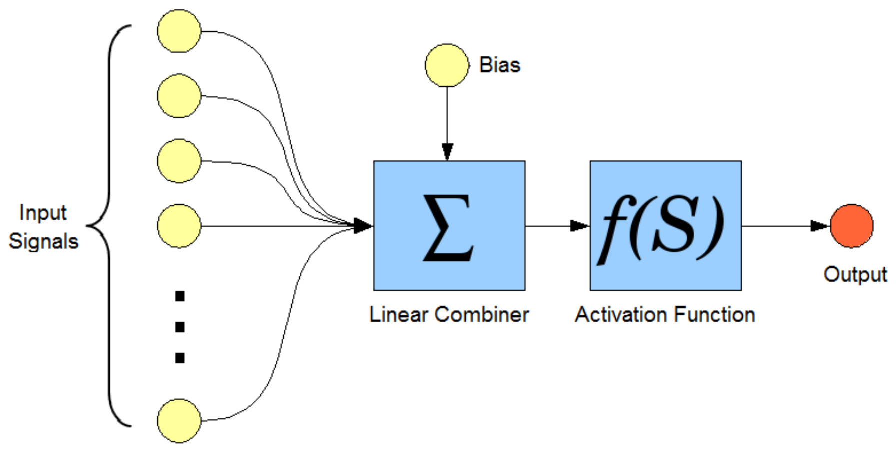

Figure 2.2: Schematic structure of an AN

Figure 2.2 illustrates the schematic structure of an AN (Engelbrecht, 2007; Haykin, 1999). Apart from the three basic elements described above, there is an externally applied bias, which adjusts the net output of the linear combiner. In mathematical terms, the function of an AN can be expressed as follows:

$$y = f(\sum_i x_i w_i + b) \tag{2.1}$$

where x_i represents the *ith* input signal from the connected neurons, w_i is the corresponding weight assigned to that neuron, b is the bias, f is the activation function and y is the output of the AN.

2.2.2.1 Activation Functions

The activation function is a key element in the ANs as it characterises the behaviours (linear, non-linear, discrete, etc.) of ANs. There exist several types of activation functions. Four of the commonly used activation functions are illustrated below (Zhang, 2009).

1. Linear function. This produces a linearly modulated output and the mathematical expression of this function is:

$$a = f(n) = \lambda n \tag{2.2}$$

where a is the output, λ is the slope of the function, which is generally set to 1, and n is the input. Figure 2.3(a) illustrates this function.

2. Step function, also known as the hard limit function. This is a binary output function producing one of the two scalar output values depending on the value of the threshold θ. The mathematical expression of the function is:

$$a = f(n) = \begin{cases} a_1 & if\ n \geq \theta \\ a_2 & if\ n < \theta \end{cases} \tag{2.3}$$

where θ is the threshold value and a_1 and a_2 are the two scalar values. Usually, the binary output would be either of the pairs (0, 1) or (-1, 1). Figure 2.3(b) illustrates this function.

3. Sigmoid function. This function is one of the most commonly used activation functions, whose graph is s-shaped. It is defined as a strictly increasing function. The expression is:

$$a = f(n) = \frac{1}{1+e^{-\lambda n}} \tag{2.4}$$

where λ is the slope parameter, which controls the steepness of the function. Figure 2.3(c) illustrates this function.

4. Hyperbolic tangent function, also known as tangent sigmoid function. The graph of this function is s-shaped and the output of the function is in the range (-1, 1). The expression of the function is:

$$a = f(n) = \frac{e^{\lambda n}-e^{-\lambda n}}{e^{\lambda n}+e^{-\lambda n}} = 1 - \frac{2}{\mathrm{e}^{2\mathrm{x}}+1} \tag{2.5}$$

where λ is the slope parameter. Figure 2.3(d) illustrates this function.

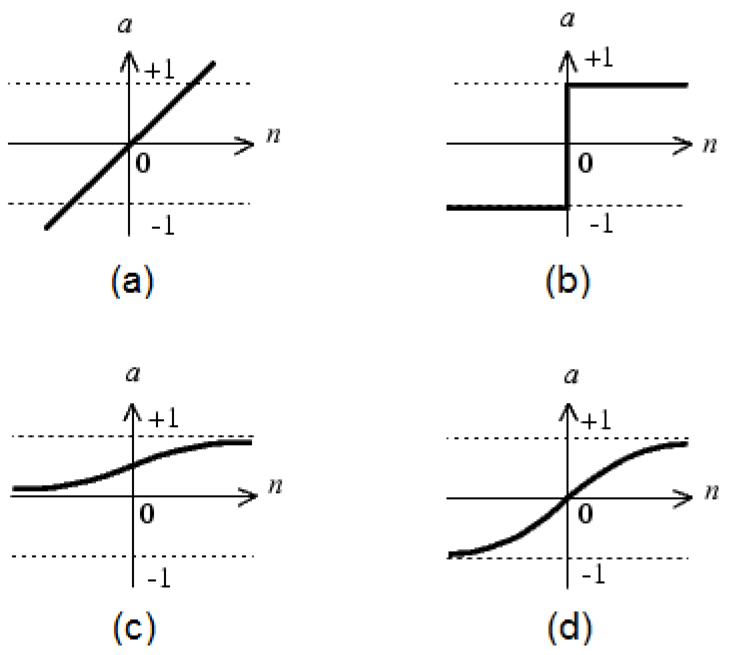

Figure 2.3: Activation functions. (a) linear function; (b) step function; (c) sigmoid function; (d) hyperbolic tangent function.

2.2.3 Fundamental Artificial Neural Networks

Similar to biological neural networks, ANs have to be connected in certain ways to perform specific tasks. The connected groups of ANs are known as ANNs. Each AN in the ANN receives inputs from either the external environment or another connected AN and generates the global output of the ANN or provides an input signal to another neuron. Depending on the structure of the inter-connections and the way to adjust or train the strengths of the inter-connection between the ANs, ANNs can be classified into various categories as described in the following section.

2.2.3.1 Structural Categorisation

In terms of their structures, ANNs can be divided into three categories, which are Feedforward Neural Networks (FNNs), Recurrent Neural Networks (RNNs) and Self-Organising Maps (SOMs) (Haykin, 1999; Haykin, 2009).

In an FNN, the neurons are grouped into various layers and the connections are based on the neurons in different layers. Signals flow from the input layer to the output layer through various numbers of layers, known as the hidden layers. The connections in FNNs are unidirectional. An FNN may be considered as a hierarchical system, in which the input layers are placed at the bottom layer, the output layers are the topmost layers, and the hidden layers are placed between the input and output layers. The ANs are connected from one layer to the next layer above it and there is no connection between the ANs in the same layer. Figure 2.4 illustrates the structure of a simple multi-layer FNN.

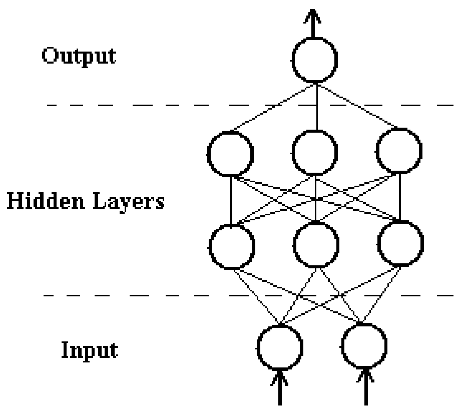

Figure 2.4: A simple multi-layer FNN

Popular FNNs include Multi-Layer Perceptrons (MLPs) and Radial Basis Function Networks (RBFNs). MLPs are the most popular category of ANNs and are widely used in various applications. They have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner (Haykin, 1999; Haykin, 2009), in which the expected outcomes are provided along with the inputs. RBFNs have similar structures to MLPs. But instead of applying linear or sigmoid activation functions, neurons in RBFNs use radial basis functions (RBFs) as their activation functions. Typical RBFs include the Gaussian function and the Multiquadratic function (Rao and Srinivas, 2003). The expression of the Gaussian function is:

$$f(r) = \exp\left(-\frac{r^2}{\delta^2}\right) \qquad \sigma > 0 \tag{2.6}$$

and the expression of the multiquadratic function is:

$$f(r) = \sqrt{r^2 + \sigma^2} \qquad \sigma > 0 \tag{2.7}$$

MLPs and RBFNs are examples of non-linear layered feedforward neural networks and they are both universal approximators.

In a RNN, the connections to a layer of neurons are not only from the adjacent layer below, but also from some ANs from the same layer or the layers above. The connections from the same layer or the layers above are generally called feedback connections, which add the ability to learn from the temporal characteristics of the input signals as they can 'memorise' the previous states of the network (Pham and Liu, 1995). Figure 2.5 illustrates the structure of a simple RNN.

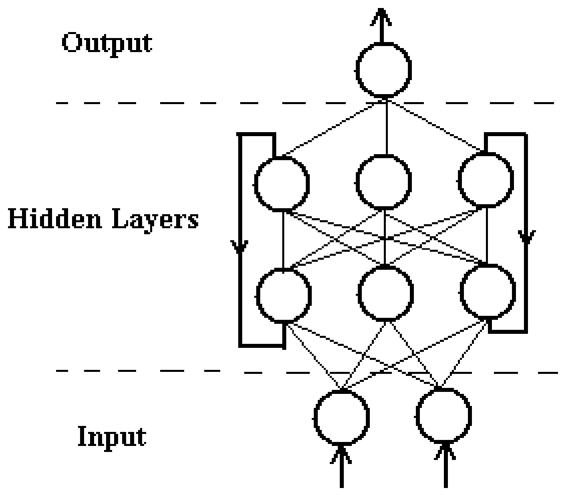

Figure 2.5: A simple RNN

SOM is a feed forward neural network using unsupervised learning algorithm (Hassoun, 1995; Kohonen, 2001). In SOMs, neurons are placed at the node of a lattice, which usually contains one or two dimensions. Figure 2.6 illustrates a simple SOM network. Neurons in SOMs have two different types of connections, which are forward connections and lateral connections. The forward connections connect the neurons in the input layer to the neurons in the output layer. The lateral connections are the connections between the neurons in the output layer.

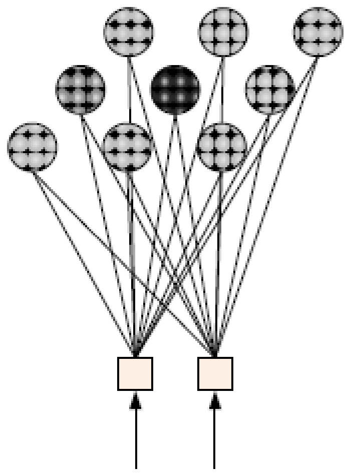

Figure 2.6: A simple SOM (adopted from Yang, 2010)

2.2.3.2 Learning Algorithm Categorisation

As the ANNs learn from the external environment, there are two types of learning algorithms allowing them to adapt to the environment: supervised learning and unsupervised learning. In addition to these two learning algorithms, there is a third type, the reinforcement learning, which is a special form of supervised learning (Basheer and Hajmeer, 2000; Nilsson, 1998).

Supervised learning, also known as associative learning, requires a train set consisting of an input vector and it corresponding output vector. The output vector is used to determine how well the network is trained to adapt to the environment and the strengths or weights of the inter-neuron connections are adjusted according to the difference between the provided output vector and the actual network output vector to reduce the overall error. Among the ANNs mentioned in the previous section, FNNs and RNNs use supervised learning algorithms.

Unsupervised learning aims to discover patterns or features in the input data without 'knowledge' from the environment. It does not require the expected output vector to be provided. During training, only the input vectors are passed to the neural network and the strengths or weights of the inter-neuron connections are adjusted automatically to group the input vectors into different clusters. The SOM network is a good example of unsupervised learning.

Reinforcement learning, also known as graded training, is defined as learning by trial-and-error from feedback from the environment. Instead of providing desired outputs, a reinforcement learning algorithm evaluates the goodness of the network output corresponding to a given input rather than an explicit target to be replicated. Depending on the performance of the network, a reward, which can be a positive or negative signal, will be issued. The reward causes a change in the network and therefore affects the future performance of the network.

2.2.4 Applications and Advantages of ANNs

It has been decades since the first introduction of ANNs. Thousands of researchers have put their efforts into improving the ANN models and applying ANNs to various applications. Nowadays, ANNs have been shown to be useful in a wide variety of practical applications and their potential is far from realisation. Some common applications of ANNs include modelling, data analysis, forecasting and optimisation in areas such as speech recognition, pattern recognition and classification, image processing, and system control (Rao and Srinivas, 2003; Taylor, 1995; Taylor, 1996). The characteristics or advantages of ANNs that make them superior in these practical applications are discussed below:

- **Computational ability**. It has been proved mathematically by Cybenko (1989) that MLPs can be used as universal function approximators. MLPs can virtually approximate any function with any level of desired accuracy if there are sufficient numbers of ANs in the hidden layer and the amount of available data points is sufficient.
- **Non-linearity**. Depending on the type of activation function used in the ANs, ANNs can behave in either linear or non-linear ways. Non-linear systems have inputs that are not proportional to the outputs. The non-

linearity of ANNs allows them to learn the nonlinear relationships within the training data directly. Traditional linear models are simply inadequate when modelling non-linear data.

- **Flexibility/adaptability**. This is the key characteristic that makes ANNs 'intelligent'. ANNs can learn from the external environment by using examples, which are represented by a set of training data. ANNs adapt to the environment by automatically adjusting the internal parameters, which are the strengths or weights of the connections between ANs. ANNs generalises 'knowledge' to produce adequate responses to unknown situations based on the 'relationship' discovered in the previous examples.
- **Parallel processing**. Neurons in the ANN are individual processing units and are typically placed in parallel structures. The computations in ANNs may be carried out in parallel as well depending on the structures of the ANNs. The parallel processing allows the computation to be performed more rapidly and special hardware implementations are being designed to take advantage of this feature.
- **Robustness and fault tolerance**. ANNs are noise-insensitive and capable of handling incomplete data (Basheer and Hajmeer, 2000). As ANNs are distributed information systems and each AN is an arithmetic element, the parallel processing ability makes ANNs relatively fault tolerant. The failure of one or more ANs or connections in the network may degrade the performance and accuracy of the system but it does not break the entire network (Du and Swamy, 2006; Li, 1994).

2.2.5 Disadvantages of ANNs

Although ANNs are undoubtedly powerful tools for many applications, they do have some possible weaknesses in them.

- ANNs do not produce explicit models. ANNs are regarded as 'black box' processing tools. Apart from defining the structure of an ANN and perhaps the initial weights of the connections, there is no other activity to interact with the network than to feed it with the available data. In the general applications of ANNs, no information is available in the form of symbolic expressions representing the behaviours of the ANNs and the relationship between the inputs and outputs. The network itself is the expression of such a relationship.
- ANNs are generally lack the means to explain. It is usually difficult to justify the correctness of a result as the connection strengths/weights do not have obvious interpretations.
- There is no structured methodology for choosing the appropriate network topology, determining effective initial internal parameters, training the neural network, and verifying the network. It is known that the performance of an ANN is generally determined by a number of factors, including number of network layers, number of neurons in each layer, the connections topology, the type of activation functions of each neuron, the training process, etc. The selection of these determinant parameters is another research area. In general, developers would conduct several trials to identify an appropriate configuration of the network (Li, 1994; Shriver, 1988).

2.3 Evolutionary Computation (EC)

EC is a key subfield of AI that may generally be used to solve combinational optimisation problems. The term 'evolution' usually refers to the optimisation process that aims to improve the ability of a species or system to survive in a competitive environment. A French biologist, Jean-Baptiste Lamarck (1744-1829), defines evolution as heredity, which is the inheritance of acquired traits, in his theory of evolution. His main idea is that individuals adapt to the environment during their lifetimes and pass the traits that make them survive to their offspring. The offspring then continue to adapt. Charles Darwin (1809-1882), whose theory of natural selection became the foundation of biological evolution, states in his theory of evolution that in a world of limited resources, each individual has to compete with others in order to survive. Those individuals with the best traits, which make them outstanding, are more likely to survive and reproduce, and those traits will be passed on to their offspring (Affenzeller, 2009; Banzhaf, 1998; Engelbrecht, 2007).

EC represents a powerful search and optimisation system that employ the computational models of evolutionary processes, such as natural selection and reproduction, as the fundamental elements of the system. Evolutionary Algorithms (EAs) is a subset of EC and generally refer to the generic population based metaheuristic optimisation algorithms. Generic components in EAs include an encoding method, a fitness function, an initialisation function, a selection function and a reproduction function. The functionalities of these components are (Affenzeller, 2009; Rothlauf, 2006):

Encoding. This converts the potential solutions into the data format that can be interpreted by the EA. In the biological world, the traits of individuals are stored in their chromosomes. Chromosomes are structures of compact intertwined molecules of DNA in the nucleus of organic cells. Each chromosome contains a large number of genes, which are the units of heredity. In the context of EA, each individual is a candidate solution to an optimisation problem. The characteristics or traits of an individual are represented by a chromosome and the variables that need to be optimised are referred as genes, the fundamental units of information.

Fitness function. In the Darwinian model of evolution, individuals with the best traits are more likely to survive and reproduce. The fitness function is a mathematical function that is used to evaluate all the potential solutions represented as chromosomes and determine the ability of each individual to survive. Generally, the fitness function would issue a quantitative fitness score to each candidate to quantify how well the candidate fits the environment, or its ability to survive.

Initialisation. As EAs are stochastic population-based search algorithms, each EA contains a group of candidate solutions, known as a population. Initialisation is the first step in the EA process and it generates the initial population. The initial population is generally generated randomly to ensure a uniform representation of the entire search space. The size of the initial population has consequences in the computational complexity and exploration ability. A Large population size generally increases diversity and improves the exploration ability, but it also leads to higher computational complexity.

Reproduction. This is the process of producing offspring from selected parents using crossover and mutation operators. Crossover creates offspring by the combination/exchange of randomly selected genetic information in chromosomes from the parents. Mutation is the process of randomly changing the genetic information in chromosomes to increase genetic diversity. In general, mutation is applied as a low probability to avoid the distortion of good genetic information.

Selection. This is one of the key operators in EA. The main objective of selection is to emphasize better solutions. The selection process is usually carried out at the end of each generation to select a new population of chromosomes. The new population could be selected from only the offspring or from both the parents and offspring.

In addition to these fundamental operations, there is another key operator in EA, the **stopping condition**. The evolutionary operators are executed iteratively in an EA process until the stopping condition is met. Commonly used stopping conditions include the discovery of an acceptable solution and the maximum number of EA iterations (generations).

The different implementations of those EA components result in different EC paradigms. There are a variety of classes of evolutionary computational models, including Genetic Algorithms (GAs), Genetic Programming (GP), Evolutionary Programming (EP), etc.

2.3.1 Genetic Algorithms (GAs)

GAs represents one of the key paradigms of EA. Although the idea of GA was proposed by Fraser (1957) and later by Bremermann (1962), Holland (1975) is credited for popularising GAs and is generally considered as the father of GAs.

GAs are based on the mechanics of natural selection and natural genetic recombination using a simulated version of the survival of the fittest. The driving operators of a GA are fitness evaluation, selection and reproduction. Figure 2.7 illustrates the generic GA procedures. An initial population of a number of individuals, in the format of a selected representation scheme, is generated randomly to represent potential solutions. The quality or fitness of each individual in the initial population is then evaluated using a fitness function. A set of parent chromosomes are selected based on their fitness scores and the fittest are selected to be the parent chromosomes and are mated to produce new individuals or offspring using the genetic operators, including crossover and mutation. The offspring is believed to inherit the advantages or traits of their parents. A selection mechanism is then used to select a new group of individuals from the parents and the offspring to form the new population. Finally, all the operations performed on the initial population, including fitness evaluation, parent chromosome selection, reproduction and new population selection, are carried out iteratively until the predefined stopping condition is met.

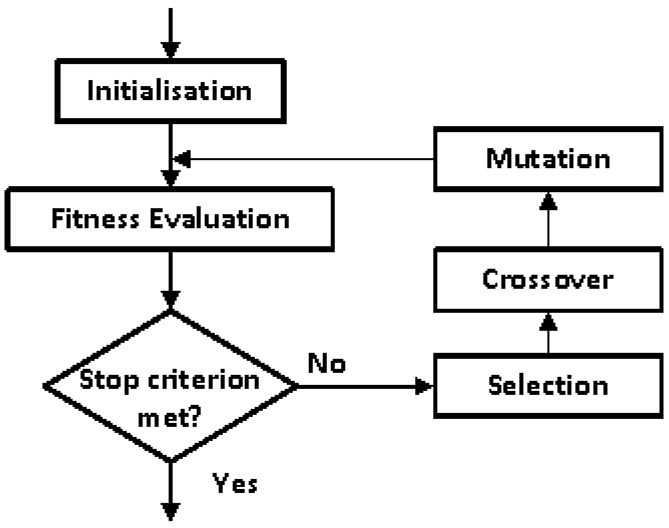

Figure 2.7: Simple GA procedures

The canonical GA proposed by Holland in 1975 follows the general procedures illustrated in Figure 2.7 and is implemented as follows:

- A bitstring (binary) of fixed length is used to represent each chromosome.
- Proportional selection is used to select the most-fit parents for reproduction.
- One-point crossover operation is used to produce offspring.
- Uniform mutation is used.

Since the introduction of canonical GA, several variations have been developed based on different mechanisms in the fundamental elements, including chromosome representation scheme, selection operator, crossover operator and mutation operator. Some of the main variants are discussed in the following sections.

2.3.1.1 Binary GA vs. Continuous GA

In general, there are two common types of chromosome representation schemes, which are the binary representation and continuous representation (floating-point representation). In the binary representation, each chromosome is represented by a string of binary numbers (0 or 1), while in the continuous representation each chromosome is a list of floating-point numbers. Figure 2.8 illustrates the difference between the binary chromosome and the continuous chromosome.

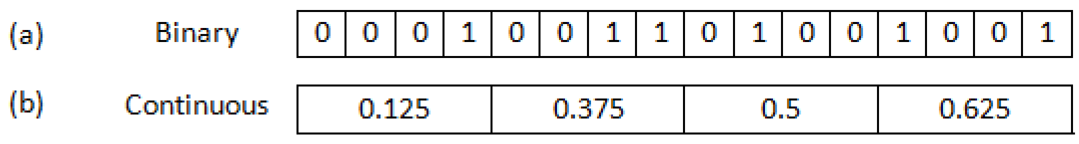

Figure 2.8: Chromosome types. (a) binary chromosome, (b) continuous chromosome.

Both the binary GA and the continuous GA follow the general processing procedures illustrated in figure 2.7. However, the binary GA requires an extra procedure, decoding, before evaluating the candidate chromosomes. The decoding procedure converts the chromosomes into variables of various formats (depending on the type of application) that can be processed by the fitness function.

2.3.1.2 Crossover

Most of the crossover operators for binary representations are applied to two parent chromosomes. Several crossover operators have been developed and these operators can be divided into four groups (Sumathi, 2010):

One-point crossover: this operator randomly selects a crossover point and the bitstrings after the crossover point are swapped between the two parent chromosomes. Figure 2.9 (a) illustrates the one-point crossover.

Two-point crossover: in this operation, two crossover points are selected at random and the bitstrings between these two points are swapped. Figure 2.9 (b) illustrates the two-point crossover.

N-point crossover: this operator is a generalised version of the two-point crossover. In this case, N crossover points are selected at random. These crossover points divide the chromosome pairs into N+1 sections indexed by positive integers, and the bitstring sections of even indices are swapped. Figure 2.9 (c) illustrates a four-point crossover.

Uniform crossover: this is a bit-wise crossover operator. Each bit in the bitstring has a probability to be swapped. A binary crossover mask can be used to indicate which bit of the parents should be swapped. This uniform crossover can be regarded as the generalised operator for all the crossover operators. Figure 2.9 (d) illustrates the uniform crossover.

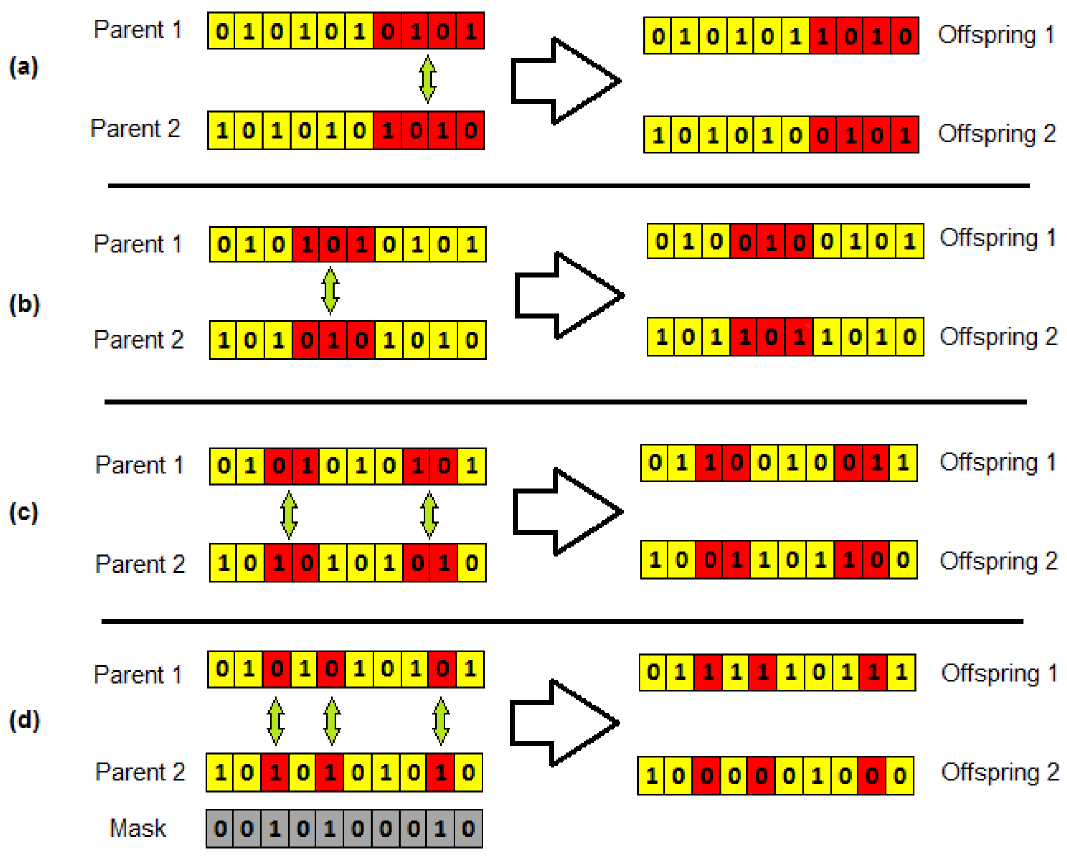

Figure 2.9: Crossover operators. (a) one-point crossover; (b) two-point crossover; (c) four-point (N-point) crossover; (d) uniform crossover.

2.3.1.3 Mutation

Mutation aims to add diversity to the genetic characteristics of the chromosomes by introducing new traits or changes to the existing individuals. It prevents the GA from converging too fast into one region. Mutation is generally applied to each gene of the offspring at a certain probability. Figure 2.10 illustrates an example of binary chromosome mutation.

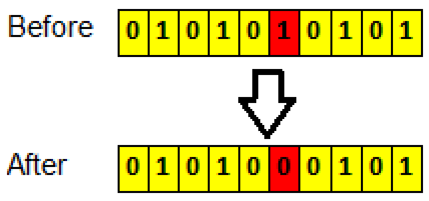

Figure 2.10: Mutation example

2.3.1.4 Advantages and Disadvantages of GAs

Some of the advantages of GAs over the conventional optimisation and search methods are:

- GAs can scan the solution hyper-space quickly and bad initial candidates (individuals) do not have a negative effect on the end solutions as they are simply discarded.
- GAs work with a coding of the parameter set, which is to be optimised, not the parameters themselves.
- GAs scan the solution hyper-space with a population of points, not a single point.

Although GAs have been proven to be an efficient and powerful optimisation and search strategy, they have drawbacks as well. The main disadvantage of GAs is actually its advantage, evolution. As evolution is inductive, it is generally slower than traditional techniques to converge to a solution and the solution is not guaranteed to be optimal but satisfactory to user's expectation.

2.3.2 Genetic Programming (GP)

GP is the extension of GA into the space of computer programs (Affenzeller, 2009). In GP, the individual chromosomes are not fixed length binary strings or numerical lists that encode possible solutions, they are programs that express the solutions to the problem. In the late 1950s, Friedberg (1958, 1959) was the first person who tried to produce computer programs by evolution. The first attempt to apply GAs to tree-structured program induction was proposed by Cramer (1985). John Koza explored the power of evolutionary program induction and established the field of GP by extensive demonstrations of using GP to solve problems (Koza, 1992). Compared with GAs, the GP is different principally in it representation but inherits most of GA's features, including crossover, mutation and selection.

The GP process is similar to the GA, it starts with an initial population of randomly generated programs (tree-structured chromosomes) composed of functions and terminals. The functions include the arithmetic operators (+, -, *, /, etc.), logical operators (AND, OR, NOT, etc.) and programming operators (IF-THEN-ELSE, etc.). The terminals are the input variables/parameters to the program. Figure 2.11 illustrates two simple GP chromosomes.

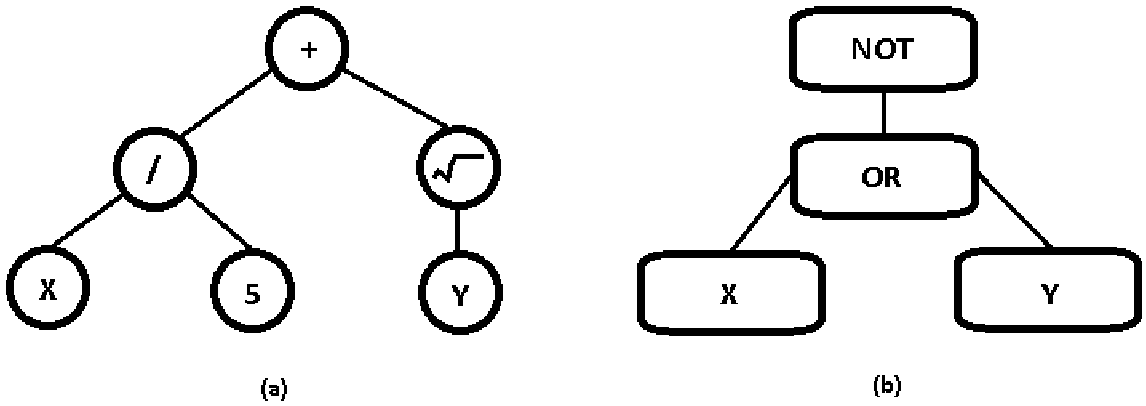

Figure 2.11: GP chromosome examples

Each GP chromosome in the population is measured in terms of how well it performs by executing the expression/program represented by the chromosome and a fitness value indicating its performance will be associated with the chromosome. After evaluating all chromosomes in the population, a new population will be created using the GP operators, including selection, crossover and mutation. The selection operator is adopted from traditional GAs, it selects pairs of chromosomes (parents) to produce new chromosomes (offsprings). Commonly used selection methods include roulette-wheel selection, random selection, stochastic uniform selection and tournament selection (Banzhaf et al., 1998; Haupt and Haupt, 2004). After selecting the parent chromosomes, the offspinrgs are produced using the crossover and mutation operators. Crossover and mutation operate on randomly selected subtrees. Figures 2.12 and 2.13 illustrate GP crossover and mutation.

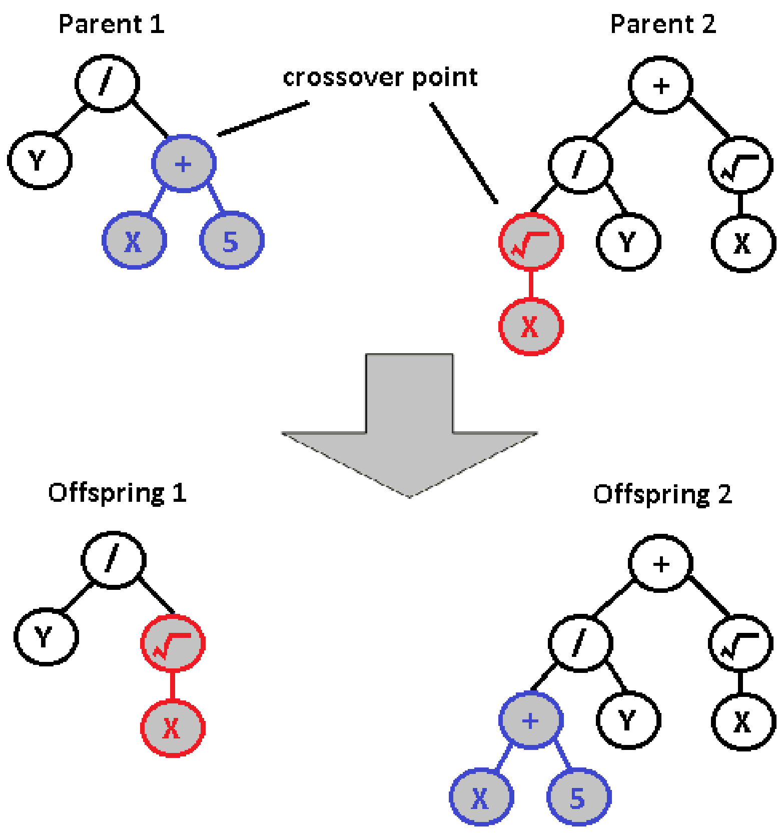

Figure 2.12: GP crossover operation

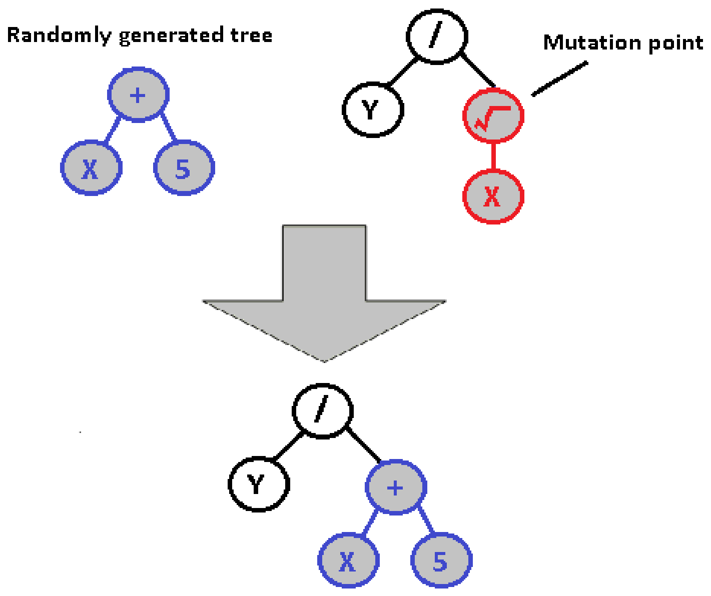

Figure 2.13: GP mutation operation

In general, a small proportion of chromosomes in the population, which have the highest fitness, are passed to the new generation without modification (crossover and mutation). The new population usually has the same number of chromosomes as the previous population and the chromosomes are evaluated followed by selection, crossover and mutation. These processes (evaluation, selection, crossover and mutation) are carried out iteratively to evolve towards a solution. However, due to the stochastic features in GP, an acceptable solution cannot be guaranteed. Thus, the stopping conditions are usually set to prevent GP from endless evolutions. Commonly used stopping conditions include the discovery of an acceptable solution, a maximum number of GP generations have been evolved and a maximum amount of processing time has been allowed. Once the GP process stops, the chromosome of the best performance discovered so far is generally considered the solution or an approximate solution to the problem.

GP has been applied successfully in many areas, such as circuit design (Lee and zhang, 2000; Wang et al., 2007), economics (Potwin et al., 2004; Salcedo-Sanz et al., 2005), chemistry (Chen et al., 2004), and symbolic regression (Salhi et al., 1998).

2.3.3 Genetic Algorithm vs. Genetic Programming

As GA and GP are both evolutionary algorithms and GP is a derivation of GA, they share many common characteristics, such as random hyper-space exploration, stochastic evolution, long processing time, etc. The main characteristics that differ them from each other are listed below (Banzhaf et al., 1998; Koza, 1992):

- **Design objective**: the GA was designed to evolve optimal numerical solutions while GP was designed to evolve expressions or programs.
- **Representation**: in GA, chromosomes take various forms, such as bit strings or numeric lists. In GP, chromosomes are executable structures, such as computer programs or symbolic expressions.
- **Fitness evaluation**: in GA, the fitness of a chromosome is usually derived by evaluating the chromosome using a predefined fitness function and the chromosome itself is either the input to the fitness function or the representation which is used to produce inputs to the fitness function. In GP, the fitness is usually assessed by executing the chromosome and each chromosome can be considered as a fitness function.

2.3.4 Genetic Neural Mathematical Method (GNMM)

GNMM is a general purpose hybrid intelligent optimisation technique developed by Jianhua Yang in 2007 (Yang, 2010; Yang et al., 2007). GNMM inherits the key characteristics of GA and ANNs, such as robustness and nonlinearity, and it is usually used as a pattern classifier and analyser. In addition, GNMM incorporates GA and mathematical programming to perform feature selection and rule extraction respectively (Yang, 2010). Yang (2007, 2008 and 2010) has demonstrated the utility of GNMM in many applications, such as predicting the longitudinal dispersion coefficient in rivers, optimising the number of e-nose sensors, selecting effective channels in EEG signals, and more.

GNMM is usually performed in three steps (Yang, 2010; Yang et al., 2007; Yang et al., 2008):

1. **GA based input variable selection**. In this step, GA is used to evolve an optimal set of variables from the available variables. The selected variables are used as the inputs in MLP models, which are also used as the fitness functions in the GA. An adaptive mutation rate is used in the GA based on the average fitness of successive generations. When the former generation has higher fitness, the mutation rate is reduced to encourage exploitation of the current discoveries; conversely, when the fitness is lower, the mutation rate is increased to encourage further exploration in a wider search space. GNMM uses the elite group and appearance percentages of individual variables to minimize the randomness associated with GA. Instead of using the chromosome of the best performance discovered in GA as the final solution, GNMM uses the appearance percentages of the

variables in the winning chromosome (the best performing chromosome) in each generation as indications of the variables' preferences. Variables of higher appearance percentages are more preferable than those with lower appearance percentages.

2. **MLP modelling**. In this step, MLPs are used as the tools to perform the actual tasks, which are classification or regression. The input variables of these MLPs are the groups of variables of high appearance percentages selected from the results produced in the first step. The common MLP training problem, overfitting, is avoided by K-fold cross-validation. The fast Levenberg-Marquardt (LM) learning algorithm is used to achieve a second-order speedup in the MLP training.
3. **Rule extraction using mathematical programming**. In this step, the mathematical programming technique proposed by Tsaih and Lin (2004) is implemented to extract regression rules from the trained MLPs. This mathematical method is not only used to identify the polynomial regression rules, but is also used to explore features from the extracted rules based on data samples associated with each rule.

2.3.5 Sensitive Genetic Neural Optimisation (SGNO)

SGNO performs dimension reduction by selecting input variables relevant to the neural modelling of the input-output relationship using ANNs, GAs and Sensitivity Analysis. The curse of dimensionality is a serious issue on models of multivariate data processing because there are many more possible combinations of variables that can be observed in datasets. As the data dimensionality increases, the complexity of data processing increases, in some cases even exponentially. In case of ANN applications, additional input variables require extra processing (learning and testing) time and computer resources. However, extra input variables cannot guarantee better performance (better fitting) as the extra inputs may bring extra noise into the neural model and the learning process cannot eliminate the noise in dataset. In addition, an ANN treats all input variables equally and it is unable to determine which variable or group of variables from all input variables can produce the best fitting.

GAs had been approved to be a good tool in FS. However, GA's application is limited to the variable combinations generated during the GA evolution and the size of the best combination is unpredictable. In case a fixed number of variables are preferred, a common approach is to find all variable combinations generated during the evolution that meet the size requirement and use the combination giving the best fitness. If the situation changed and more variables are required instead of the previous group of variables, a completely new variable group needs to be discovered. This approach is unfavourable for the applications where reorganising the whole input configuration is impractical (expensive sensors or complicated systems). SGNO simplifies this process by selecting variables based on their importance in the system and extra variables can be identified by looking up in the variable importance rank table.

2.4 Conclusion

In this chapter, some of the well-known ISTs, including ANNs, GAs and GPs are reviewed briefly. In addition, a recently developed hybrid intelligent optimisation technique, GNMM, is introduced as well. These techniques are powerful tools that have been proved to be successful in many areas, such as machine learning, decision support, pattern recognition, and data regression.

Among these techniques, ANNs are often referred to as 'black box' models as there is generally no interactive activity with an ANN apart from feeding it with the available data. In addition, ANNs are generally lack of explanation capabilities and there is no information available to represent their behaviours and the relationship between the inputs and outputs. GAs and GPs are known as evolutionary algorithms due their biological background, which is natural evolution and selection. Due to their differences in solution representation and evolution operations, GAs are well suited to perform feature selection, while GPs are more suitable for rule extraction and decision making. GNMM is implemented using ANNs and GAs, it inherits the advantages from these two techniques. As a result, it is able to optimise the number of inputs and perform data regression and classification.

In the following chapter 3, a novel hybrid intelligent system, known as SGNO, is introduced. SGNO is a general purpose optimisation (feature selection) tool that combines the advantages of ANNs and GAs together with the ability to quantify the influence/importance of individual variables on the model output.

References

Affenzeller, M. (2009). *Genetic algorithms and genetic programming: Modern concepts and practical applications*. Boca Raton: CRC Press.

Banzhaf, W., Nordin, P., Keller, R. E., & Francone, F. D. (1998). *Genetic Programming, An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*: Morgan Kaufmann.

Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods, 43*, 3-31.

Beale, R., & Jackson, T. (1990). *Neural Computing: An Introduction*. Bristol: Adam Hilger.

Berthold, M., & Hand, D. J. (2007). *Intelligent data analysis: an introduction*. Berlin: Springer.

Bremermann, H. J. (1962). Optimisation through Evolution and Recombination. In M. C. Yovits, G. T. Jacobi & G. D. Goldstine (Eds.), *Self-Organisation Systems* (pp. 93-106): Spartan Books.

Chen, X., Gui, W., Wang, Y., & Cen, L. (2004). Multi-step optimal control of complex process: a genetic programming strategy and its application. *Engineering Applications of Artificial Intelligence, 17*(5), 491-500.

Cramer, N. L. (1985). *A representation for the adaptive generation of simple sequential programs.* Paper presented at the International Conference on Genetic Algorithm and their Applications, Carnegie-Mellon University.

Cybenko, G. (1989). Approximations by superpositions of sigmoidal functions. *Mahtematics of Control, Signals and Systems, 2*, 303-314.

Du, K. L., & Swamy, M. N. S. (2006). *Neural Network in a Softcomputing Framework*. London: Springer.

Engelbrecht, A. P. (2007). *Computational Intelligence: An Introduction* (2nd ed.). Chichester: Wiley.

Fraser, A. S. (1957). Simulation of Genetic Systems by Automatic Digital Computers I: Introduction. *Austrilian Journal of Biological Science, 10*, 484-491.

Fraser, A. S. (1957). Simulation of Genetic Systems by Automatic Digital Computers II: Effects of Linkage on Rates of Advance Under Selection. *Austrilian Journal of Biological Science, 10*, 492-499.

Friedberg, R. M. (1958). A Learning Machine: Part I. *IBM Journal of Research and Developmenet, 2*, 2-13.

Friedberg, R. M. (1959). A Learning Machine: Part II. *IBM Journal of Research and Developmenet, 3*, 183-191.

Haken, H. (1996). *Principles of Brain Functioning: A Synergetic Approach to Brain Activity, Behavior and Cognition*. Berlin: Springer.

Hassoun, M. H. (1995). *Fundamentals of Artificial Neural Networks*: The MIT Press.

Haupt, R. L., & Haupt, S. E. (2004). *Practical Genetic Algorithms*. New Jersey: Wiley-Interscience.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation* (2nd ed.). New Jersey: Prentice-Hall.

Haykin, S. (2009). *Neural Networks and Learning Machines* (3rd ed.). New Jersey: Pearson Education.

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

Kohonen, T. (2001). *Self-Organizing Maps* (3rd ed.). Heidelberg: Springer.

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

Lee, K. J., & zhang, B. T. (2000). *Learning robot behaviors by evolving genetic programs*. Paper presented at the 26th annual conference of the IEEE.

Li, E. Y. (1994). Artificial neural networks and their business applications. *Information & Management, 27*, 303-313.

Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis*. San Francisco: Morgan Kaufmann Publishers.

Parks, R. W., Levine, D. S., & Long, D. L. (Eds.). (1998). *Fundamentals of Neural Network Modeling*: The MIT Press.

Pham, D. T., & Liu, X. (1995). *Neural Networks for Identification, Prediction and Control*. Tyne and Wear: Springer.

Potwin, J. Y., Soriano, P., & Vallee, M. (2004). Generating trading rules on the stock markets with genetic programming. *Computers and Operations Research, 31*(7), 1033-1047.

Rao, M. A., & Srinivas, J. (2003). *Neural Networks: Algorithms and Applications*. Pangbourne: Alpha Science.

Richards, D., Clark, T., & Clarke, C. (Eds.). (2007). *The Human Brain and its Disorders*. Oxford: Oxford University Press.

Salcedo-Sanz, S., Fernandez-Villacanas, J. L., Segovia-Vargas, M. J., & Bousono-Calzon, C. (2005). Genetic programming for the prediction of insolvency in non-life insurance company. *Computers and Operations Research, 32*(4), 749-765.

Salhi, A., Glaser, H., & De Roure, D. (1998). Parallel implementation of a genetic programming based tool for symbolic regression. *Information Processing Letters, 66*(6), 299-307.

Shriver, B. (1988). Artificial Neural Systems. *IEEE Computer, 12*(3), 8-9.

Sumathi, S. (2010). *Computational Intelligence Paradigms: theory & applications using Matlab*. Boca Raton: CRC Press.

Taylor, J. G. (1995). *Neural Networks*. Oxon: Alfred Waller.

Taylor, J. G. (Ed.). (1996). *Neural Networks and Their Applications*. London: John Wiley and Sons.

Tsaih, R., & Lin, C.-C. (2004). The Layered Feed-Forward Neural Networks and Its Rule Extraction. In F.-L. Yin, J. Wang & C. Guo (Eds.), *Advances in Neural Networks – ISNN 2004* (Vol. 3173, pp. 377-382): Springer Berlin / Heidelberg.

Wang, F., Li, Y. X., Li, L., & Li, K. (2007). Automated analog circuit design using two-layer genetic programming. *Applied Mathematics and Computation, 185*(2), 1087-1097.

Yang, J. H. (2010). *Intelligent data mining using artificial neural networks and genetic algorithms: techniques and applications.* Unpublished PhD, University of Warwick, Coventry.

Yang, J. H., Hines, E. L., Iliescu, D. D., & Leeson, M. S. (2007). *GNMM and accurate longitudinal dispersion coefficient prediction*. Paper presented at the The Seventh UK Chinese Association of Resource and Environment (CARE) Annual Meeting.

Yang, J. H., Hines, E. L., Iliescu, D. D., Leeson, M. S., & Boilot, P. (2008). Optimising the Number of Electronic Nose Sensors needed using Genetic Algorithms and Neural Networks. In E. L. Hines, M. S. Leeson, M. Martinez-Ramon, M. Pardo, E. Llobet, D. D. Iliescu & J. H. Yang (Eds.), *Intelligent Systems: Techniques and Applications*: Shaker Publishing.

CHAPTER 3

Sensitive Genetic Neural Optimisation

3.1 Overview

Chapter 2 has provided the theoretical background to some important ISTs. As explained in section 2.3.1, a Genetic Algorithm (GA) is a stochastic heuristic tool capable of generating useful global solutions to optimisation and search problem. An Artificial Neural Network (ANN), discussed in section 2.2.3, is a non-linear adaptive modelling tool which is usually used to model complex relationships. This chapter introduces a hybrid optimisation system, namely the Sensitive Genetic Neural Optimisation (SGNO) system based on GA, ANN and sensitivity analysis (SA). The system is designed to identify the most efficient subset of input variables from all the available ones, also known as dimensionality reduction.

One obvious advantage of dimensionality reduction is that it reduces the amount of data and thus simplifies the problem. In term of ANNs, reducing dimensionality (number of variables) reduces the number of input parameters of the network. The selection of appropriate subset of variables is important to obtain good generalisation for data driven techniques like ANNs with finite data (Bishop, 2005; Tarassenko, 1998). The dimensionality reduction is usually achieved either by selecting features from the raw data, commonly known as feature selection (FS) or deriving new features from the raw data, known as feature derivation (FD) (Marsland, 2009). Broadly speaking, the SGNO is an intelligent FS algorithm.

The SGNO system combines the advantages of GA, which are the stochastic search in the search space and the evolution towards better solutions, together with the strength of SA in the study of how the model responds to the changes in the inputs (Saltelli et al., 2000). The SA provides further and detailed study on the search spaces that have been explored by the GA. In this thesis, the terms ANNs and Neural Networks (NNs) are used interchangeably. Figure 3.1 illustrates the general structure of this optimisation system.

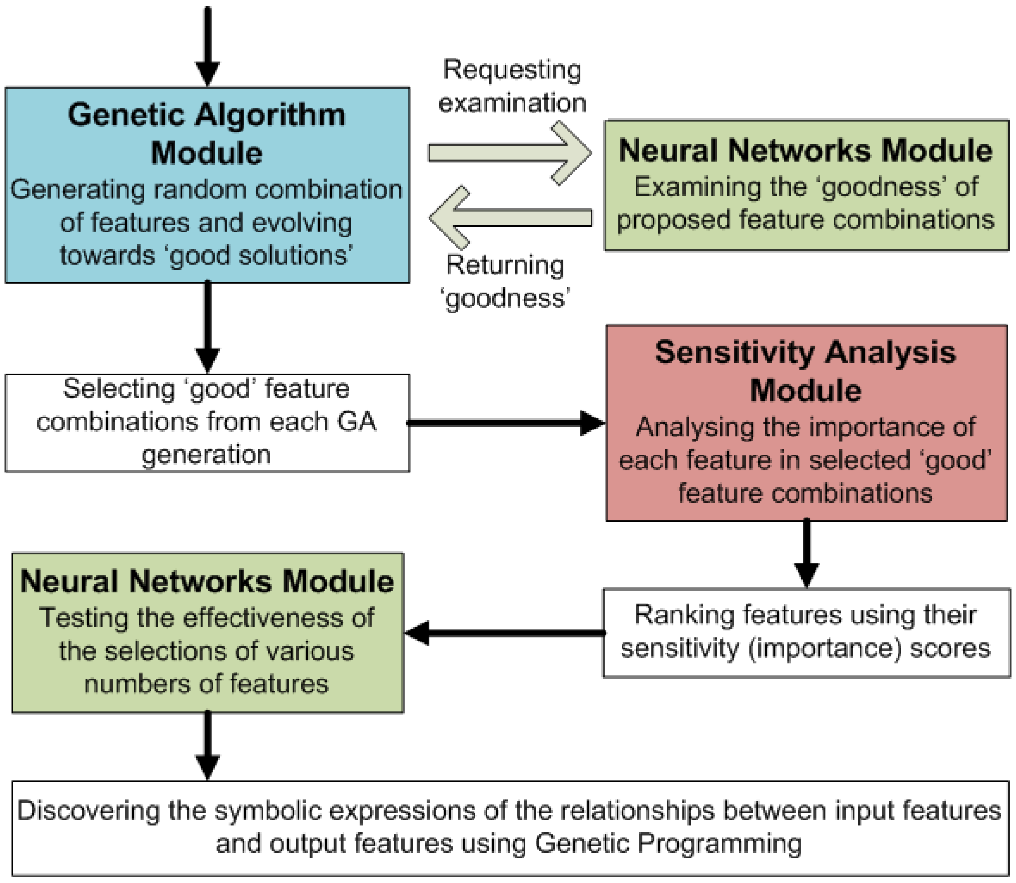
Genetic Algorithm Module
Generating random combination of features and evolving towards 'good solutions'
Requesting examination
Returning 'goodness'
Neural Networks Module
Examining the 'goodness' of proposed feature combinations
Selecting 'good' feature combinations from each GA generation
Sensitivity Analysis Module
Analysing the importance of each feature in selected 'good' feature combinations
Ranking features using their sensitivity (importance) scores
Neural Networks Module
Testing the effectiveness of the selections of various numbers of features
Discovering the symbolic expressions of the relationships between input features and output features using Genetic Programming

Figure 3.1: General structure of the SGNO system

The SGNO system can be divided into three parts, the GA module, the SA module and the NN module. The GA module defines the key skeleton of the entire system, which is the same as the structure of a standard GA optimisation system. The NN module plays the parts of (a) the fitness function in the GA module and (b) the final modelling tool after the most efficient input subset is selected. Depending on the design of the NN module, the SGNO can be used to optimise and solve either regression modelling or classification problems. The SA module refines the stochastic search in the search space by investigating how the NN model responds to the variations in its inputs and thus identifying the variables that contribute the most to the output variability. The input variables that contribute more variation to the outputs are believed to be more influential than those variables that generate less variation in the model and thus the model is more sensitive to these variables (Saltelli et al., 2000).

3.2 SGNO Explained

The optimisation procedures of SGNO consist of three key steps including data pre-processing, variable selection (optimisation) and system remodelling. The details of SGNO are explained in the following sections.

3.2.1 Data Pre-processing

In practice, ANNs rarely operate on raw data. The raw data is usually processed using initial pre-processing procedure(s) that transforms the raw data into reasonably comparable ranges or eliminates some of the raw data that carry redundant information (Tarassenko, 1998). For most applications, data transformation is necessary as the raw data usually contains noisy variables presented in different ranges. For ANNs, transforming the variables in the raw data into similar or specific ranges allows the ANNs to learn more quickly and perform better (Bishop, 2005).

In the simplest case, data transformation takes the form of a linear transformation, such as Min-Max normalisation, expressed as function 3.1, and Mean-Standard Deviation normalisation, expressed as function 3.2.

$$\boldsymbol{X'} = \frac{\boldsymbol{X} - \mathbf{min}\,(\boldsymbol{X})}{\mathbf{max}(\boldsymbol{X}) - \mathbf{min}\,(\boldsymbol{X})} \tag{3.1}$$

where *X* represents the samples of a variable in the raw data and *X'* represents the transformed variable.

$$\boldsymbol{X'} = \frac{\boldsymbol{X} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{3.2}$$

where *X* represents the samples of a variable in the raw data, μ is the mean of *X* and σ represents the standard deviation of *X*.

3.2.2 Variable Optimisation

Variable optimisation is the most important step in SGNO, it employs three techniques, which are GA, ANN and SA. These techniques are known as the core modules in SGNO. The GA module controls the progress of the algorithm and employs the NN module as its fitness function. The SA module quantifies the importance of each available variable using the results generated in the GA module. The main procedures in this variable optimisation step include variable encoding, initial population generation, population evaluation, new population generation and variable selection. Each is now considered in turn.

3.2.2.1 Variable Encoding

In the theory of evolutionary biology, evolution takes place by operations on chromosomes, which hold the characteristics of the living subjects (Sivanandam and Deepa, 2008). In GAs, the potential solutions are encoded as chromosomes, which may be represented (encoded) in different formats depending on the type of application. General encoding methods include bit strings and real numbers (Reeves and Rowe, 2003). Real number encoding represents chromosomes as a series of real numbers and is usually used when the GA is used to search for a numerical solution. Bit string encoding, also known as binary encoding, represents chromosomes as collections of binary digits (1 and 0) and it is commonly used to solve optimisation and FS problems (Handels et al., 1999; Lu et al., 2008).

In this case in SGNO, binary encoding is more appropriate as the binary values (0 and 1) are efficient and effective to determine the absence or presence of a certain variable. The length of a chromosome, i.e. the number of binary digits in the bit string, corresponds to the number of potential variables. Each bit in the chromosome represents the absence or presence of a specific variable. In general, 1 indicates that the variable is selected (present) and 0 indicates that the variable is not selected (absent).

3.2.2.2 Initial Population Generation

A GA begins with a population of potential solutions encoded in the chromosomes, known as the initial population. The initial population is generally generated randomly across the search space. The size of the initial population is crucial in a GA. A small number of chromosomes may only search a small region of the solution space and the GA may take a very long time to find a reasonable solution and it is likely to lock on a local minimum or maximum. A large population of chromosomes covers too much of the search space. It often lacks genetic diversity and may require a large number of generations to find high performance chromosomes (Cox, 2005). The population size is generally dependent on the complexity of the problem and there is no broadly agreed 'optimal' number (Cox, 2005; Sumathi, 2010). Sivanandam and Deepa (2008) summarised that a population of around 100 chromosomes is frequently used. Sumathi (2010) suggests that a larger number would be useful but demands excessive costs of memory and time. In SGNO, a rule of thumb suggested by Cox (2005), which calculates the minimum size of initial populations based on the number of variables and states, is employed. The formula is expressed in 3.3 and says that the size should be at least as large as five times the number of variables or half the number of possible states, whichever is smaller.

$$\boldsymbol{p = min\left(5v, \frac{1}{2}\, s\right)} \qquad (\mathbf{3}.3)$$

where *p* is the estimate of the population size, *v* is the number of variables and *s* is the number of possible states in the search space.

For example, a GA containing 10 variables in binary encoding has 2^{10}=1024 possible states and thus the suggested initial population size should be at least

$$\boldsymbol{p = min\left(5 \times 10, \frac{1}{2} \times 1024\right) = min(50, 512) = 50}$$

3.2.2.3 Population Evaluation

Each chromosome in the population is evaluated by a fitness function and a goodness-of-fit or performance measure is associated with the chromosome. In SGNO, Multi-Layer Perceptrons (MLPs) are used to determine the fitness of chromosomes. Instead of generating goodness-of-fit value, MLPs produce an error measure for each chromosome. The details of MLPs are discussed in the following sub-sections.

3.2.2.3.1 MLP Architecture

As discussed in section 2.2.5, the characteristic features of an MLP include the number of inputs and outputs, the number of hidden layers, the number of hidden neurons in each hidden layer, the weights of the neurons and the activation functions for the neurons.

In SGNO, the number of inputs is determined by the number of ON-bits (digit 1) in the binary chromosome and the number of outputs depends on the application. In regression problems, e.g. the applications presented in chapters 4 to 7, there is usually only one output. For some classification problems, the number of outputs may be more than one and each output indicates the association with a certain type of class/group. The number of hidden layers is generally difficult to determine a priori. The literature suggests that one hidden layer is adequate to approximate any function with arbitrary accuracy (Ghaffari et al., 2006; Moghaddam et al., 2010; Tarassenko, 1998). Therefore, MLPs of one hidden layer are used.

The number of neurons in the hidden layer is very import and hard to determine. It affects the training time and generalisation of the MLP. A large number of hidden neurons may allow the ANN to memorise, also known as overfit, the pattern it has been trained with, whereas too few hidden neurons may waste a great deal of training time in order to try to generalise. There is no general rule for determining the optimal number of hidden neurons. The most popular approach to finding the optimal number of hidden neurons is by trial and error (Ahmed, 2005), which is an experimental method of reaching a satisfactory solution by trying out various means until error is eliminated or sufficiently reduced. However, a trial and error approach is impractical in SGNO as it increases the complexity of the system dramatically. Therefore, SGNO employs a simple rule of thumb approach, which dynamically determines the numbers of hidden neurons based on the numbers of inputs and outputs instead of assigning a fixed number of neurons to all ANNs. It halves the sum of the number of inputs and outputs as expressed in equation 3.4.

$$\boldsymbol{H} = \left\lceil \frac{\boldsymbol{1}}{\boldsymbol{2}} (\boldsymbol{N_{in}} + \boldsymbol{N_{out}}) \right\rceil \tag{3.4}$$

where $\lceil \quad \rceil$ is the ceiling operator, and N_{in} and N_{out} are the numbers of inputs and outputs respectively.

The activation functions of the neurons are determined based on the type of the application. In general, sigmoid functions are commonly used in forecasting and regression, while step functions are used for classification. Details of activation functions are discussed in section 2.2.2.1. The commonly used weights initialisation method, which generates random numbers between -1 and 1, is employed to initialise the neurons in MLPs (Sumathi, 2010).

3.2.2.3.2 Chromosome Evaluation

Evaluation of a chromosome involves training the MLP with a certain subset of the available data and testing the trained MLP with the rest of the data, which has not been 'seen' by the MLP. However, overfitting may occur if the MLP is trained too many times, which leads to poor generalisation and test results. To overcome this problem, SGNO performs multi-fold cross-validation with early stopping.

To realise this, the data is randomly partitioned into 5 approximately equal-sized groups and these data groups are used for training (3 groups), validation (1 group) and testing (1 group) in turn (MathWorks, 2010; Setiono, 2001). Figure 3.1 illustrates the five-fold cross-validation.

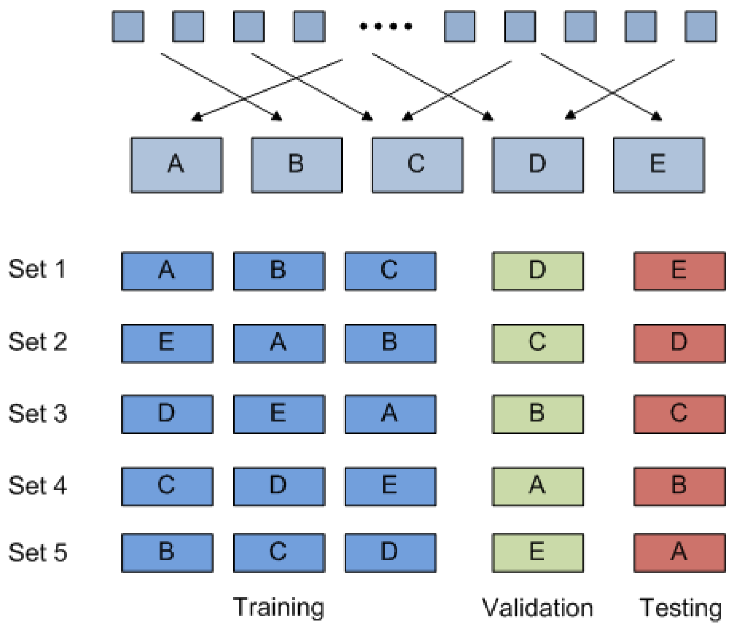

Figure 3.2: Five-fold cross-validation

The training subset is used to train the NN, i.e. updating the weights in neurons. The validation subset is used to determine when to stop training (early stopping) and it does not contribute to the weights update in neurons. The testing subset is used to measure the performance of the trained NN by evaluating the data in the subset.

During the training process, the trained NN is monitored continually by keeping a close watch on the validation error. Experience suggests that the training error and validation error decrease at the early stage of training. After a certain number of training cycles, the training error still decreases while the validation error stops decreasing or even starts to rise. It is a good indicator as the start of overfitting (over training) and training should stop. Hence the stopping criterion should be the point at which the minimum validation error is reached (Prechelt, 1998; Tarassenko, 1998). Figure 3.2 illustrates the early stopping scenario.

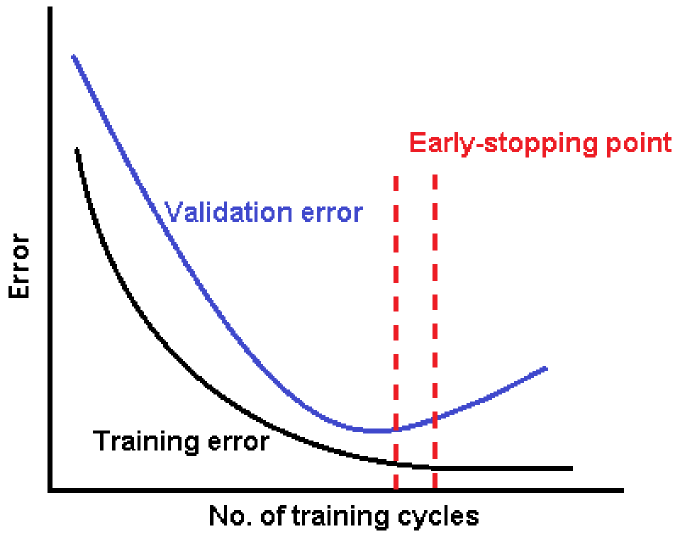

Figure 3.3: Early stopping scenario

Finally, the performance of the trained NN is measured using the test subset. The NN takes the input variables from the testing subset and generates outputs. The mean squared error (MSE) between these generated outputs and the corresponding expected outputs in the testing subset are then calculated as the assessment of fitness.

$$\boldsymbol{MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - y'_i)^2} \qquad (\mathbf{3}.5)$$

where N is the number of input-output pair in the test subset, y_i represents the output generated by the NN and y'_i is the corresponding expected output.

In the case of five-fold cross-validation, five MLPs are required, one for each of the possible sequential combinations of the training, validation and testing groups. The mean of the MSEs from the five MLPs is determined and used as the fitness of the chromosome.

3.2.2.4 New Population Generation

After evaluating all the chromosomes in the current generation, a new generation of chromosomes will be generated based on the fitness values using genetic operators, such as selection, crossover and mutation. Regeneration and evaluation are repeated until a predefined termination criterion is met. Commonly used termination criteria include fixed maximal number of evolved generations or the attainment of an acceptable level of fitness (Sumathi, 2010).

Selection is an important operator in GAs, it chooses the parents from the chromosomes in the current generation based on their fitness to produce offspring chromosomes in the new generation. Common selection methods include stochastic uniform selection and roulette wheel selection. In SGNO, stochastic uniform selection is used as it provides zero bias, which means an individual's selection probability equals its expected number of trials, and minimum spread, which is the minimum number of trials in selection that theoretically permits zero bias (Baker, 1987; Zalzala and Fleming, 1997). It lays out a line in which each chromosome in the generation corresponds to a portion of the line and the length of that portion is proportional to the scaled fitness value. A pointer scans along the line in equal steps. At each step, the chromosome corresponding to the section it lands on is selected as a parent. Chromosomes with higher fitness values have better chances to be selected. Figure 3.3 illustrates stochastic uniform selection.

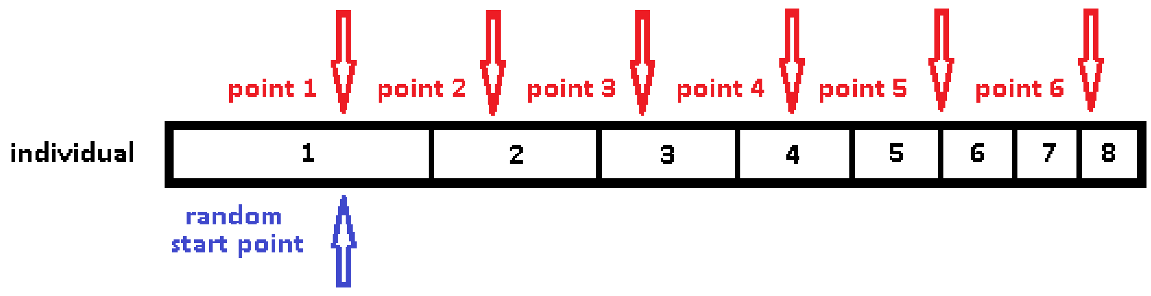

Figure 3.4: Stochastic uniform selection operation

Parent chromosomes will then be used to generate offspring. There are four common crossover methods as discussed in section 2.3.1.2. In SGNO, the N-point (scattered) crossover is used as it is the most flexible crossover operator and all other crossover operators can be considered as special cases of this operator. New chromosomes are generated by swapping the bits at a number of random positions of the parent chromosomes. Figure 3.5 illustrates the operation of N-point crossover.

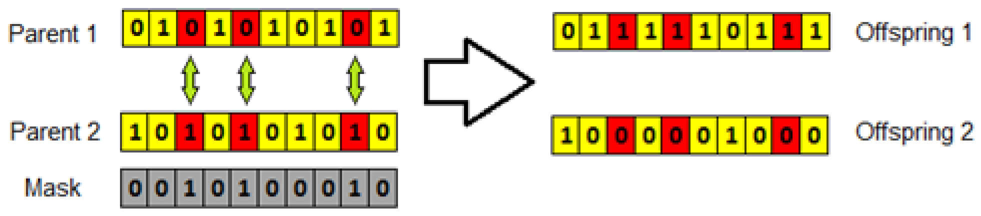

Figure 3.5: N-point (scattered) crossover

In general, a small number of chromosomes with the highest fitness in the generation are guaranteed to be passed to the next generation without modification to ensure the best characteristics are 'preserved' in the new generation. These are called the elite chromosomes or elite children (Haupt and Haupt, 2004; Sumathi, 2010).

3.2.2.5 Variable Selection

After the termination of the GA process, a large number of chromosomes and their fitness for the problem will be available. Among them, those chromosomes with fitness in the upper quartile range in a generation are selected to carry out detailed study, the SA analysis, of the importance of the variables in those chromosomes.

3.2.2.5.1 Sensitivity Analysis (SA) of High Fitness Chromosomes

The SA is a group of statistical evaluation methods in mathematical modelling. It helps to determine how the changes in the values of parameters influence the outputs of the model, i.e. how 'sensitive' the model is to changes in parameters.

The sensitivity of an input variable or parameter is an indication of the effect that a variation of that input will have on the output. An input variable of higher sensitivity will result in a greater variation of the output and vice versa. In general, the variables of higher sensitivity would draw higher level of interest and thus are more favourable than those of lower sensitivity (Saltelli et al., 2000; Saltelli et al., 2008).

In SGNO, the sensitivity of an input variable is defined as its impact on the outputs when it is replaced by arbitrary values instead of fixed values at a number of discrete levels as in the original SA. The SA analysis is implemented using the Monte Carlo Simulation (MCS).

3.2.2.5.2 Monte Carlo Simulation (MCS)

The MCS is a powerful stochastic technique which has been applied in many fields of mathematics, physics and engineering. It is a method that uses random processes or random numbers and performs statistical sampling experiments to find approximate solutions to problems. The key characteristic of MCS is the simulation process based on random numbers (Dimovivan, 2008; Shonkwiler and Mendivil, 2009).

The MCS is usually used to simulate complicated problems which are difficult and time-consuming to analyse or the problems have some components behaving in a random way. In SGNO, the MCS used to estimate the sensitivity of a particular variable in a given chromosome is summarised as follows:

Step 1: for a chromosome of *M* selected variables, a group of *M+1* random numbers is generated and each random number is within the domain of its corresponding variable in the chromosome. The extra random number represents an arbitrary value of the variable of interest and it is within the domain of that variable.

Step 2: the group of random numbers is evaluated using the following equation and the generated value is stored at the end of a numeric list.

$$I_i = \left| \frac{Y(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_M) - Y(x_1, x_2, \ldots, x_{i-1}, x_{M+1}, x_{i+1}, \ldots, x_M)}{x_{M+1} - x_i} \right| \tag{3.6}$$

where $x_1, x_2, \ldots, x_M$ represent the random numbers for those variables in the chromosome, x_i is the variable of interest, x_{M+1} is the extra random number to replace x_i, and *Y* represents the trained NN, which has *M* input variables.

Step 3: repeat step 1 and step 2 for a fixed number of cycles, say 10 or 20, and then calculate the mean of the values stored in the numeric list.

Step 4: repeat all the previous steps until the mean values converge. Then the latest mean value, which is calculated using a large number (hundreds or thousands) of I_i, is used as an approximation of the sensitivity of the *i*th variable.

These 4 steps are repeated to approximate the sensitivities of all the variables in the chromosome. Hence the variables of higher sensitivity scores are of higher importance as they generate higher variation in the outputs.

3.2.2.5.3 Variable Ranking

By repeating the MCS described in the previous section, 3.2.2.5.2, a sensitivity matrix containing all the sensitivity scores of all the selected chromosomes is produced. The rows in the sensitivity matrix represent those selected chromosomes and each column holds the sensitivities of a particular variable in different chromosomes. The global sensitivity score (the mean sensitivity score) of a variable will be calculated by taking the mean of its sensitivity scores in all chromosomes. This can be expressed as the following equation:

$$S_i^* = \frac{1}{N}\sum_{j=1}^{N} S_{j,i} \tag{3.7}$$

where S^*_i is the global sensitivity score of the *i*th variable, N is the number of chromosomes in the sensitivity matrix and $S_{j,i}$ is the sensitivity score of the *i*th variable in the *j*th chromosome.

Based on the global sensitivity scores, the variables can be ranked. The variables with higher global sensitivity scores are preferable over those of lower scores as they are considered more important/influential to the model. It is possible that some of the variables are missing, i.e. they are not present in the selected chromosomes. These variables will not be considered as they are ‘eliminated’ by the GA process.

3.2.3 System Remodelling using MLP

In SGNO, MLPs are not only the evaluation functions during the GA process but they are also used as the modelling tool to generate the efficient solutions using the optimised variables. Following the determination of variables’ importance, the performance of SGNO is evaluated by MLPs taking various numbers of input variables with the highest global sensitivity scores. The architectures of the MLPs used in this evaluation are estimated in the same way as the MLPs used as the fitness functions in the GA process (see section 3.2.2.3.1 for details).

As discussed previously in section, 2.2.5, the key features that affect the performance of MLPs include the number of hidden layers, the number of neurons in each hidden layer, the initial weights and activation functions of the neurons, the number of training cycles and the training functions.

It is well-known that an MLP without a hidden layer is only capable of representing linearly separable functions or decisions; one hidden layer is generally sufficient for approximating any functions containing continuous mapping from one finite space to another; an extra hidden layer helps to model any functions of any shape including discontinuities. In addition, there is no theoretical reason to use MLPs of more than two hidden layers (Heaton, 2005).

There is also no clear determination of the numbers of hidden neurons to be used in an MLP of best performance. However there are many rules-of-thumb methods to approximate the number of hidden neurons. Some of them are summarised as follows (Heaton, 2005):

- The number of hidden neurons should be between the input layer size and the output layer size.

$$N_{outputs} < N_{hidden} < N_{inputs}$$

- The number of hidden neurons should be two thirds of the input layer size plus the size of the output layer.

$$N_{hidden} = \frac{2}{3}N_{inputs} + N_{outputs}$$

- The number of hidden neurons should be no more than twice the input layer size.

$$N_{hidden} \leq 2N_{inputs}$$

These rules-of-thumb methods are guidance only. To find the most appropriate architecture for a particular problem, the selection is normally made by trial and error (Heaton, 2005). In SGNO, the architecture of the best performance is achieved by applying a 'forward' selection which is an iterative operation that gradually increases the number of hidden neurons in the MLP until either an acceptable error level is achieved or increasing the number of neurons would not improve the performance further. Again, the five-fold cross-validation with early stopping criterion and random initial weights in the neurons are applied (Setiono, 2001).

3.2.4 Symbolic Rule Discovery using Genetic Programming

Genetic Programming (GP) is an extension of GA, differing principally in its operators and representations. In GP, the individuals (chromosomes) are expressions, such as a piece of computer program or an equation, instead of binary strings. One of the advantages of Genetic Programming (GP) is the tree-structured representation of chromosomes, which is particularly useful to generate tree-based solutions, such as mathematical expressions and decision trees. In SGNO, a specially tuned GP method using Multiple-Branch Encoding (MBE) is employed to find an appropriate and concise mathematical expression for the best performing NN discovered. The key features of the GP method are discussed in the following subsections.

3.2.4.1 Multiple-Branches Encoding

MBE was proposed by Rodríguez-Vázquez and Oliver-Morales (2003) and aims to produce simple syntax expressions of good quality avoiding the problem of oversized expressions, which is also known as bloat (Rodríguez-Vázquez and Oliver-Morales, 2003; Rodríguez-Vázquez and Oliver-Morales, 2004). In MBE, instead of using a single tree structure as a chromosome, the weighted sum of a group of single trees (multiple branches) and a constant value is used as the MBE chromosome. Figure 3.4 illustrate the construction of an MBE chromosome.

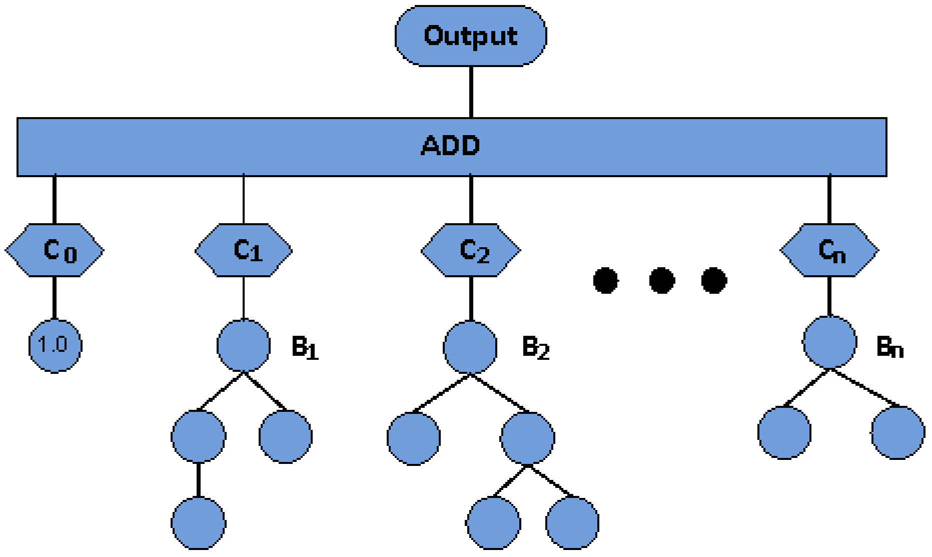

Figure 3.6: MBE chromosome example

The mathematical expression for the MBE chromosome is as follows:

$$Y = C_0 + \sum_n B_i C_i \tag{3.8}$$

$$B_i = F_i(X) \tag{3.9}$$

where Y represents the output, C_0 is the constant value, B_i is the ith branch, C_i is the weight of the ith branch, F_i is the function expressed in the ith branch, and X represents all the inputs.

The weight for each branch is determined using multiple linear regression (MLR) based on the set of training data and the performance (fitness) of the MBE chromosome is measured using the test data set.

Compared with the conventional GPs, MBE-GP can be more effective theoretically to deliver an acceptable solution due to the encoding method. Instead of relying on the proposed individual chromosome, MBE-GP establishes connections (the weight compositions) between tree-structured branches, which are used as individual chromosomes in conventional GPs. Thus, MBE-GP is more likely to find a solution.

3.2.4.2 MBE Population Initialisation

In this MBE-GP, a syntax tree is constructed using input variables and operators in an operator pool, i.e. +, –, *, /, etc. Instead of using MBE chromosomes of fixed length (the number of trees in a chromosome) proposed by Rodríguez-Vázquez and Oliver-Morales (2003), the length of a chromosome is dynamic up to an upper limit. The depth of the trees in chromosomes is limited to 3 in the initial population i.e. there are at most three operators and four terminals (variables) in a tree.

It is not hard to imagine that the problems using GP are generally more complicated than those applying GA. The GA based optimisation using binary string representations attempts to find the 'optimal' combination of binary digits in chromosomes, while GP chromosomes consists not only of the input variables, but also the available mathematical operators and the syntax trees that represent the potential relationships. Hence, the population size of GP is considerable, usually several hundred or thousand (Iba et al., 2009; Poli et al., 2008).

Similar to the GA process discussed previously in section 3.2.2, the initial population of this MBE-GP is generated randomly. Again, the estimate of population size is a problem. There is no agreed way to estimate an appropriate number and researchers choose population sizes up to several thousand at their own wills (Koza, 1992; Xie et al., 2007). In the MBE-GP proposed here, the population size is determined using an extension of Cox's formula for GA population size estimation (see equation 3.3). Thus, the population size is determined using the following expression:

$$p = 5 * v * op \tag{3.10}$$

where p is the population size, v is the number of inputs (terminals) and op is the number of operators.

After generating the initial population, all chromosomes will be evaluated and a new generation will be produced using the evaluation results.

3.2.4.3 Chromosome Evaluation

As expressed in Equations 3.8 and 3.9, each chromosome represents a linear composition of a group of functions, which can be linear or non-linear. The coefficient (weight) of each function is not determined when the chromosome is produced. Hence the evaluation of a chromosome takes two steps, which are coefficient determination using MLR, and testing. Five-fold cross-validation is used in this evaluation as well. Instead of dividing the data set into three groups, which are training (60%), validation (20%) and testing (20%), GP chromosome evaluation requires only two groups, which are training (80%) and testing (20%). The training data set is used to determine the coefficients in the chromosome and the testing data set is used evaluation its performance.

3.2.4.4 New Population Generation

The main genetic operators, crossover and mutation, in MBE-GP are defined in a similar way as those in GA. The crossover between two MBE chromosomes is performed on the basis of branches, a subset of trees (branches) in each chromosome is randomly selected and the selected branches are crossed over in the traditional way. The mutation operation can be performed at any level (individual branch or sub-branch). The selected tree/sub-tree in a chromosome would be replaced by a newly generated tree/sub-tree up to the current maximum depth of trees (Ghanea-Hercock, 2003; Rodríguez-Vázquez and Oliver-Morales, 2003).

In this MBE-GP, there are two different methods to generate the new population, i.e. low level generation and high level generation. Low level generation is similar to the generation method used in GAs as discussed in section 3.2.2.4. It generates offspring by applying crossover and mutation operators to the chromosomes selected using stochastic uniform selection (Banzhaf et al., 1998; Koza, 1992). In addition, a small group of chromosomes of high fitness (elite group) is passed to the new generation without modification.

The role of high level generation is to increase the complexity (depth of the trees) of the chromosomes every time it is applied. This should only be applied if an acceptable solution has not been discovered after a certain number of generations, say 50. High level generation selects a relatively large portion of the chromosomes, say 30%, of high fitness in the populations that have been explored so far and passes them to the new population without modification. New chromosomes with higher complexity are generated to fill the new population, e.g. assuming the current tree depth limit is 4 and the new chromosomes would contain branches up to 5 levels.

These two generation methods work together to gradually expend the search space and look for the ‘optimal’ solution. The stopping criterion of this MBE-GP would be the discovery of an acceptable solution, or that a certain number of generations have been evaluated, or the chromosomes reach a certain level of complexity (overcomplicated), in which case the MBE-GP fails to derive a solution. In the case that an acceptable solution cannot be found after a few trials, modifications can be made to the population size and the complexity of branches to cover a larger search space. Figure 3.5 illustrates the general operations of the MBE-GP.

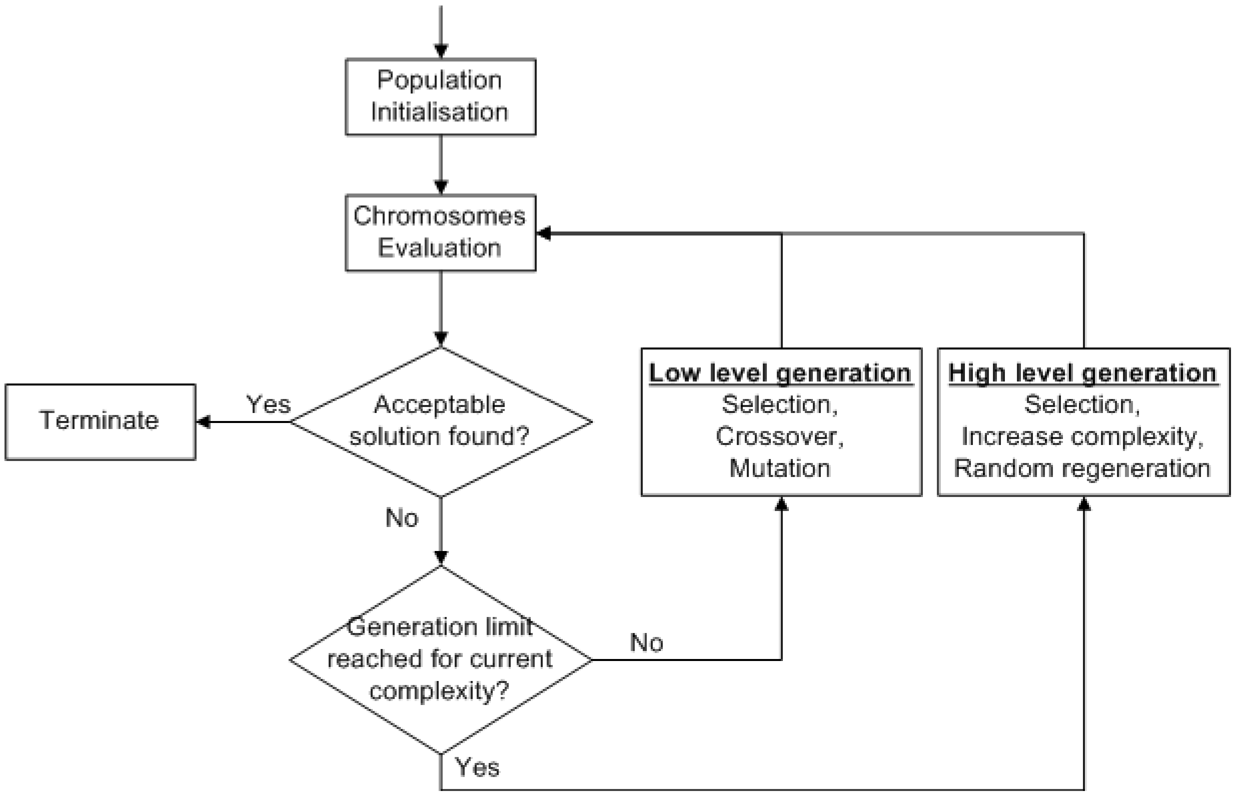
Population Initialisation
Chromosomes Evaluation
Acceptable solution found?
Yes
Terminate
No
Generation limit reached for current complexity?
No
Yes
Low level generation
Selection, Crossover, Mutation
High level generation
Selection, Increase complexity, Random regeneration

Figure 3.7: MBE-GP structure

3.3 Benchmarking Techniques

In order to assess the relative performance of SGNO, several commonly used dimensionality reduction techniques are implemented in this thesis to benchmark the SGNO. The selected benchmarking techniques include Principal Component Analysis (PCA), Forward Feature Selection (FFS), Backward Feature Selection (BFS) and the Genetic Neural Mathematical Method (GNMM). The following sections explain these benchmarking techniques in turn.

3.3.1 Principal Component Analysis (PCA)

PCA is one of the most famous mathematical techniques widely used in data dimensionality reduction. It performs vector space transformation on the given data set to rearrange the data into a new coordinate system (Dunteman, 1989). The transformed data is the linear combinations of the original data. In the new coordinate system, the number of coordinates is generally equals to the original coordinate system and all coordinates are orthogonal to each other. The new coordinates are ordered so that the first coordinate accounts for most of the variations in the original data; the second coordinate explains the maximum variances for the residual data; the third coordinate explains the majority of the variation for the next residual data and so on. Hence, the first coordinate is considered to be the most important coordinate, known as the first principal component (PC); the second coordinate is called the second PC, and so on. Generally speaking, a certain number of PCs, less than the number of coordinates in the original data, are enough to account for most of the variance in the original data. This is to say that PCA can transform a high dimensional data set to a lower dimensional space without losing significant amounts of information when compared with the original data set (Dunteman, 1989; Zhang et al., 2006).

In general, a PCA transformation can be performed using the following 4 steps (assuming X is an *n-by-p* matrix, where n is the number of variables and p is number of records in each variable):

Step 1: subtract the mean value

$$X' = X - \bar{X} \tag{3.11}$$

Where $\bar{X}$ is an *n-by-1* vector containing the mean of each variable in data set X, and X' is the new data set.

Step 2: calculate the covariance matrix

$$A = cov(X') = (X'X'^T)/(p-1) \tag{3.12}$$

Where S represents the covariance matrix, X'^T is the transpose of matrix X' and p is the size of each variable.

Step 3: calculate the eigenvectors and the eigenvalues of the covariance matrix

$$Ax = \lambda I x \tag{3.13}$$

$$(A - \lambda I)x = 0 \tag{3.14}$$

$$\det(A - \lambda I) = 0 \tag{3.15}$$

Where A is the square covariance matrix derive in step 2, I is the identity matrix, λ is a vector of eigenvalues and x is the matrix of eigenvectors.

Step 4: reorder the eigenvectors based on their associated eigenvalues, from the highest to the lowest, which represent the explained variance of the eigenvectors.

3.3.2 Forward Feature Selection (FFS) and Backward Feature Selection (BFS)

FFS and BFS are two widely used conventional FS techniques. These are also known as the sequential methods as they both select features/variables one by one. FFS starts with an empty selection list and repetitively adds features to the list. At each step, all the variables that are not in the list are tested individually with the variables already in the list and the one that generates the lowest error is added to the list. This process is repeated until all features are included in the list or when certain predefined conditions (stopping criteria) are met, such as acceptable level of accuracy or limitation on the number of selected variables. In the case where all the variables are added to the list, the order in which the variables are added to the list could be considered to be the preference order of the variables and the variables added at earlier times are more preferable than the latter ones. Table 3.1 illustrates the general routine of the FFS (Rub and Kruse, 2010).

Table 3.1: FFS algorithm

```
S = [ ]                        {solution list}
F = features                   {available features list}
 REPEAT
    E = [ ]                    {temporatory evaluation results}
    FOR j = 1 TO length(F)
       f = F[j]                {select jth feature}
       Sj = [S, f]             {add jth feature to current solution}
       M = model(Sj)           {create a regression model}
       Ej = evaluate(M)        {evaluate the regression model}
       E = [E, Ej]             {save evaluation result}
    END FOR
  S = [S, F[min(E)]]           {add the best feature to solution list}
  F = F-F[min(E)]              {remove the best feature from features list}
END REPEAT IF min(E)<threshold OR F=[ ] OR length(S)>threshold
RETURN S
```

On the other hand, BFS tries to solve problems from another direction; it starts with a list containing all the variables and iteratively removes variables from the list. At each step, the variables in the list are tested and the one that produces the lowest error when eliminated is removed from the list. This process is repeated until the following conditions are met:

1. All variables are removed, or
2. A certain level of estimation error is reached, or
3. A certain number of variables are eliminated.

Unlike the FFS, the variables selected to be eliminated at early times in BFS are considered unfavourable. Table 3.2 illustrates the procedures in BFS (Jain and

Zongker, 1997; Karagiannopoulos et al., 2007; Pudil et al., 2002; Rub and Kruse, 2010).

Table 3.2: BFS algorithm

```
S = [ ]                        {solution list}
F = features                   {available features list}
 REPEAT
    E = [ ]                    {temporatory evaluation results}
    FOR j = 1 TO length(F)
       f = F[j]                {select jth feature}
       Fj = F - f              {remove jth feature from feature list}
       M = model(Fj)           {create a regression model}
       Ej = evaluate(M)        {evaluate the regression model}
       E = [E, Ej]             {save evaluation result}
    END FOR
  S = [F[min(E)], S]           {add the worst feature to solution list}
  F = F-F[min(E)]              {remove the worst feature from features list}
END REPEAT IF min(E)>threshold OR F=[ ] OR length(S)>threshold
RETURN S
```

In this thesis, the implementation of the FFS and BFS uses NNs as the regression models, while the FFS and BFS only set out the routine of the selection procedures. These NN models employ the same architectures as the NN module in SGNO as discussed previously in section 3.2.2.3.1, e.g. single hidden processing layer and the number of hidden neurons is estimated by halving the total number of inputs and outputs of the NN model.

3.3.3 Genetic Neural Mathematical Method (GNMM)

GNMM is a general purpose intelligent FS technique developed by Yang (2007). The technique is implemented using GAs and ANNs. In GNMM, the preferences of variables are determined by their appearance percentages during the GA process. The variables of higher appearance frequencies are more preferable than those of lower appearance frequencies. The implementation details were discussed in section 2.3.4.

3.4 Theoretical Computational Complexity

The efficiency of an algorithm is generally measured by its computational complexity. In the literature, this concerns determining an expression for the number of steps (operations) needed to finish the task as a function of the problem size. As the exact step count is generally difficult to measure and the complete expression might be hard to interpret, instead of obtaining the exact step count, one attempts only to get asymptotic bounds, denoted using the Big-O notation, on the step count (Sait and Youssef, 1999; Yang, 2008).

In the case of this work, the conventional Big-O notation, which indicates the upper bound of operational steps, is inappropriate as GAs and ANNs are both non-deterministic systems, in which the intermediate processing states and the outputs cannot be predicted. Thus, instead of representing the efficiency of an algorithm using the number of operational steps, a simple notation method, which measures the number of ANN training processes, is used to indicate the efficiency of an algorithm. This notation method is feasible as ANN is the key functioning component to evaluate the performance of the proposed solutions (subsets of input variables) in SGNO and all the benchmarking techniques, including BFS, FFS, GNMM and PCA. ANN training is probably the most time consuming operation in the implementation of all these techniques, as it involves repetitive updates of the hidden neurons' weights, which are represented as matrices in complexly structured data blocks. In addition, the number of update cycles in the training process is unpredictable (Blum and Rivest, 1992; Wang, 1995). In modern implementations of ANNs, the training process is usually limited by three conditions to ensure that it can terminate within acceptable times and thus these conditions can be considered as the upper bounds of ANNs' complexities. These conditions are (MathWorks, 2010):

1. The total number of training cycles, or
2. The detection of convergence, or
3. An acceptable level of testing error.

In BFS, FFS, GNMM, PCA and SGNO, the ANN components are of similar architectures and are trained and evaluated using the identical five-fold data set. All these ANN components only have a single hidden processing layer; the number of hidden neurons in BFS, FFS, PCA and SGNO is estimated by halving the numbers of inputs and outputs, while GNMM employs fixed number of hidden neurons, which is estimated by halving the number of all available input variables. Hence, the ANNs in these techniques are likely to have similar training complexities.

As discussed in section 3.2, SGNO employs a GA to outline the general structure of the algorithm and uses ANNs as the fitness functions to evaluate the chromosomes. Hence, the number of ANN trainings involved in the SGNO process is dependent on the number of GA chromosomes evaluated. The number of ANN trainings can be expressed as:

$$\boldsymbol{C = PopSize * GenSize = 5 * N * GenSize} \qquad (3.10)$$

where *PopSize* is the size of a GA population, *GenSize* is the number of GA generation evolved and *N* is the total number of variables to be optimised.

As GNMM was structured in the same way as SGNO, the number of ANN trainings involved is of similar level as SGNO and thus Equation 3.10 applies to GNMM.

BFS and FFS are of similar levels of complexity. Although they work towards a solution from distinct directions, the total number of ANN evaluations in these two techniques is the same. They both start with a number of ANN evaluations which equals the total number of variables in the first processing cycle; in the second processing cycle, the number of ANN evaluations is reduced by 1 as one of the variables is either removed or added to the solution list; in the third processing cycle, the number of ANN evaluations is reduce further by 1, and so on. Hence, the total number of ANN evaluations in BFS and FFS can be expressed as:

$$\boldsymbol{C = \frac{N(N+1)}{2}} \tag{3.11}$$

where *N* represents the total number of variables.

In PCA benchmarking, ANNs are only used to evaluate the performance of various numbers of PCs. Hence the total number of ANN evaluations is the same as the number of all available variables, which can be expressed as:

$$\boldsymbol{C = N} \tag{3.12}$$

Based on the derived expressions 3.10, 3.11 and 3.12, PCA is no doubt the most efficient technique; BFS and FFS are more expensive than PCA. The complexities of GNMM and SGNO are not directly comparable to BFS, FFS and PCA as their complexities are dependent on the number of generations as well. However, in the case of applications presented in Chapters 4 to 7, in which the GA usually evolves for 50 generations and the total number of variables is generally no more than 50, BFS and FFS are more efficient than GNMM and SGNO.

3.5 SGNO Computation Times

The implementation of SGNO and the examinations of its feasibility together with the benchmarking techniques were all completed using MATLAB® 2009a on a PC with an Intel® Core at 2.13 GHz, 4GB memory and a 150GB 3.5'' standard hard drive running Microsoft® Windows XP service pack 3. Table 3.3 summarises the computation times of SGNO on four different types of applications. The details of the applications will be explained in following chapters.

Table 3.3: Computations of SGNO

Application	Data Size	Number of Features	Computation Time
Tomato Yield Prediction	884	50	19.50 hrs
Longitudinal Dispersion Coefficient Prediction	127	20	2.26 hrs
Wave Overtopping Prediction	5277	14	72.52 hrs
Industrial Production Growth Prediction	493	50	22.35 hrs

Based on the numeric entries in Table 3.3, direct relationships between computation times and dataset characteristics (data size and number of features) are not observable. The trend of numeric entries in the bottom 3 rows suggests that the computation time is approximately proportional to the product of the data size and the number of features. However, the first numeric entry opposes this. Hence the actual computation time may depend on the size of the dataset, the number of features to be optimised and the complexity of the problem itself, which is hard to measure but it determines how quickly NNs can learn from the dataset.

3.6 Conclusion

In this chapter, the general procedures of SGNO have been explained in detail. The SGNO consists of three key modules, which are the GA module, the NN module and the SA module. The GA module controls the progress of the algorithm and employs the NN module as its fitness function to evaluate the performance of proposed chromosomes, which represent potential solutions. In the NN module, five-fold cross-validation with early-stopping is used in the training process to prevent the overfitting problem. After the execution of the GA module, the SA module takes a quarter of the population from each GA generation and calculates the sensitivity measures of the selected variables in those chromosomes. The global sensitivity measure of a variable is derived by taking the average of its sensitivity measures in the chromosomes. The importance or influence of a variable is determined by its global sensitivity measure and a variable with higher value is considered more important/influential than those with lower values.

Compared with 4 benchmarking techniques, which are BFS, FFS, GNMM and PCA, SGNO has a similar complexity level to GNMM in terms of the number of ANN trainings. PCA is the most efficient among these techniques. The complexities of BFS and FFS are more efficient than GNMM and SGNO in the case of applications presented in chapters 4 to 7 in this thesis.

References

Ahmed F.E. (2005) Artificial neural networks for disgnosis and survival prediction in colon cancer. Molecular Cancer 4.

Baker J.E. (1987) Reducing bias and inefficiency in the selection algorithm, 2nd International Conference on Genetic Algorithms. pp. 14-22.

Banzhaf W., Nordin P., Keller R.E., Francone F.D. (1998) Genetic Programming, An Introduction: On the Automatic Evolution of Computer Programs and Its Applications Morgan Kaufmann.

Bishop C.M. (2005) Neural Networks for Pattern Recognition Oxford University Press, New York.

Blum A.L., Rivest R.L. (1992) Training a 3-Node Neural Network is NP-Complete. Neural Networks 5:117-127.

Cox E. (2005) Fuzzy Modeling and Genetic Algorithms for Data Mining and Exporation Morgan Kaufmann, San Francisco.

Dimovivan T. (2008) Monte Carlo Methods for Applied Scientists World Scientific Publishing.

Dunteman G.H. (1989) Principal Components Analysis SAGE Publications.

Ghaffari A., Abdollahi H., Khoshayand M.R., Bozchalooi I.S., Dadgar A., Rafiee-Tehrani M. (2006) Performance comparison of neural network training algorithms in medeling of bimodal drug delivery. International Journal of Pharmaceutics 327:126-138.

Ghanea-Hercock R. (2003) Applied evolutionary algorithms in Java Springer-Verlag.

Handels H., Rob T., Kreusch J., Wolff H.H., Poppl S.J. (1999) Feature selection for optimized skin tumor recoginition using genetic algorithms. Artificial Intelligence in Medicine 16:283-297.

Haupt R.L., Haupt S.E. (2004) Practical Genetic Algorithms Wiley-Interscience, New Jersey.

Heaton J.T. (2005) Introduction to Neural Networks with Java Heaton Research.

Iba H., Hasegawa Y., Paul T.K. (2009) Applied Genetic Programming and Machine Learning CRC Press.

Jain A.K., Zongker D. (1997) Feature Selection: evaluation, application and small sample performance. IEEE Transactions PAMI 19:153-158.

Karagiannopoulos M., Anyfantis D., Kotsiantis S.B., Pintelas P.E. (2007) Feature selection for regression problems, HERCMS'07, Athens University of Economics and Business.

Koza J.R. (1992) Genetic Programming, On the Programming of Computers by Means of Natrual Selection The MIT Press.

Lu J.J., Zhao T.Z., Zhang Y.F. (2008) Feature selection based-on genetic algorithm for image annotation. Knowledge-Based Systems 21:887-891.

Marsland S. (2009) Machine Learning, An Algorithmic Perspective CRC Press.

MathWorks. (2010) MathWorks Matlab Documentation: Neural Network Toolbox, MathWorks.

Moghaddam M.G., Ahmad F.B.H., Basri M., Rahman M.B.A. (2010) Artificial neural network modeling studies to predict the yield of enzymatic synthesis of betrlinic acid ester. Electronic Journal of Biotechnology 12:1-12.

Poli R., Langdon W.B., McPhee N.F. (2008) A Field Guide to Genetic Programming Lulu.

Prechelt L. (1998) Automatic early stopping using cross validation: quantifying the criteria. Neural Networks 11:761-767.

Pudil P., Novovicova J., Somol P. (2002) Feature selection toolbox software package. Pattern Recognition Letters 23:487-492.

Reeves C.R., Rowe J.E. (2003) Genetic Algorithms: Principles and Perspectives - A Guide to GA Theory Kluwer Academic Publishers, Dordrecht.

Rodríguez-Vázquez K., Oliver-Morales C. (2003) Divide and Conquer: Genetic Programming Based on Multiple Branches Encoding, in: C. Ryan, et al. (Eds.), Genetic Programming, Springer Berlin / Heidelberg. pp. 218-228.

Rodríguez-Vázquez K., Oliver-Morales C. (2004) Multi-branches Genetic Programming as a Tool for Function Approximation, Genetic and Evolutionary Computation – GECCO 2004, Springer Berlin / Heidelberg. pp. 719-721.

Rub G., Kruse R. (2010) Feature selection for wheat yield prediction, in: M. Bramer, et al. (Eds.), Research and Development in Intelligent Systems XXVI, Springer-Verlag, London.

Sait S.M., Youssef H. (1999) Iterative Computer Algorithms with Applications in Engineering IEEE Computer Society Press.

Saltelli A., Chan K., Scott E.M. (2000) Sensitivity Analysis Wiley, Chichester.

Saltelli A., Ratto M., Andres T., Campolongo F., Cariboni J., Gatelli D., Saisana M., Tarantola S. (2008) Global Sensitivity Analysis: The Primer Wiley-Blackwell.

Setiono R. (2001) Feedforward Neural Network Construction Using Cross Validation. neural Computation 13:2865-2877.

Shonkwiler R.W., Mendivil F. (2009) Explorations in Monte Carlo Methods, Springer.

Sivanandam S.N., Deepa S.N. (2008) Introduction to Genetic Algorithms Springer, Berlin Heidelberg.

Sumathi S. (2010) Computational Intelligence Paradigms: theory & applications using Matlab CRC Press, Boca Raton.

Tarassenko L. (1998) A Guide to Neural Computing Applications Arnold, Bristol.

Wang S. (1995) The unpredictability of standard back propagation neural networks in classification applications. Management Science 41:555-559.

Xie H.Y., Zhang M.J., Andreae P. (2007) Genetic Programming for New Zealand CPI Inflation Prediction, Evolutionary Computation, 2007, CEC 2007, IEEE Congress on. pp. 2538-2545.

Yang J.H., Hines E.L., Iliescu D.D., Leeson M.S. (2007) GNMM and accurate longitudinal dispersion coefficient prediction, The Seventh UK Chinese Association of Resource and Environment (CARE) Annual Meeting, Greenwich, London.

Yang X.S. (2008) Introduction to Computational Mathematics World Seientific Publishing, Singapore.

Zalzala A.M.S., Fleming P.J. (1997) (Ed.)^(Eds.) Genetic algorithms in engineering systems, The Institution of Electrical Engineers, London. pp. Pages.

Zhang Y.X., Li H., Hou A.X., Havel J. (2006) Artificial neural networks based on principal component analysis input selection for quantification in overlapped capillary electrophoresis peaks. Chemometrics and intelligent laboratory systems 82:165-175.

CHAPTER 4

Greenhouse Tomato Weekly Yield Prediction

4.1 Overview

In the previous chapter, the procedures of SGNO were described in detail together with the explanations of the benchmarking techniques, which are BFS, FFS, GNMM and PCA. This chapter introduces a greenhouse tomato weekly yield prediction system developed using SGNO to process the environmental data, such as temperature, radiation, CO_2 concentration, vapour pressure deficit and previous yields, without relying on complex physiological models.

Both supermarkets and tomato growers require reliable supplies of high quality fruit in agreed quantities. Growers have increased fruit quality and yields in many parts of the world through the use of greenhouses where the growing/environmental conditions can be controlled and by selecting better cultivars. However, weekly yields can fluctuate and this can pose problems of both over-demand and over-production if the yield cannot be predicted accurately. In this respect growers and scientists are looking for ways to forecast tomato yield in order to plan greenhouse operations and marketing and thus reduce costs and increase profits. A large number of prediction models and prototypes have been developed in the past few decades, based on specialist knowledge of tomato physiology and growing conditions. However, they tend to deal accurately with total yields, but poorly with weekly yield fluctuations.

4.2 Background

As outlined in section 4.1 above, tomato growers are sometimes contracted to sell agreed quantities of produce to supermarkets. However, tomato yields often vary from week to week, and so the ability to accurately predict future yields would give them a competitive advantage. If a grower is forecast to have insufficient fruits in a given week they could source additional produce from elsewhere, and if they are predicted to have excess fruits they could look for alternative markets or arrange promotions (Zhang et al., 2010). As a result there has been considerable interest from growers and researchers in developing tomato yield prediction systems.

Modern plant production systems are very complex. The fruit yield as the output is determined to be the result of complex interactions of many factors. There is no doubt that the environment at a fixed period in the past is a major factor that has an effect on the present plant growth and development. Environmental conditions that influence the growth and development of tomato plants include air temperature (day and night), fruit temperature, radiation, CO_2 concentration, fruit load, nutrients, plant density and stress. Scientists have put considerable effort into the relationships between crop yields and various environmental conditions. The research work of Willits and Peet (1998) suggests that warmer conditions in the greenhouse at night can significantly improve the quality and quantity of tomatoes. The fluctuation of temperature affects mostly the time of fruit ripening and rate of fruit growth. The relationship between temperature and yield is complex and studies have shown that the sensitivity of fruits to temperature changes over time, as fruits become more sensitive to temperature as they approach maturity. This explains why raising the greenhouse temperature results in a peak in yield followed a few days later by a yield reduction (Adams and Valdes, 2002; Mulholland et al., 2003). However, although temperature fluctuations influence the weekly tomato yields in greenhouses, they do not significantly influence the overall tomato yields when compared with temperature controlled growing conditions (de Koning, 1988; de Koning, 1990). Other work showed that the primary cause of fluctuations in yield was due to the effect of temperature on fruit ripening and the variation in ripening time smoothes out the effect on the yield that the fruit set may have (Adams et al., 2001a; Adams et al., 2001b). Another important environmental factor is CO_2, which is a key source in photosynthesis. Nilsen, et al. (1983) discovered that the tomato total yield showed a significantly positive responsive to CO_2 enrichment in greenhouses provided that this is provided throughout the entire harvest period. Recent work by Li, et al. (2007) showed that tomato plants could benefit from CO_2 enrichment.

In general the prediction systems are based on the development of various types of mathematical and empirical models applied to parameters routinely monitored in a greenhouse. Mathematical model based systems consider the parameters that may influence the growth of tomato plants as the input variables and generate the estimated yields as the outputs using their uniquely developed algorithms. O'Kane (1973) stated that the amount of solar radiation received by the plant would be the main factor that affects the growth rate of plants in the greenhouse and introduced a model of accumulated useful solar radiation to estimate the development time of greenhouse tomatoes. Kano and van Bavel (1988) developed a deterministic model based on a photosynthesis equation and a carbon accumulation model. The model takes the concentration of CO_2, temperature and light level as inputs to calculate tomato yield, total dry matter and leaf area index. Their simulation results show that their model can achieve up to 69% accuracy. In 1995, Tang (1995) introduced a multiple linear regression based empirical model using cumulative light intensity and temperature. The empirical model could estimate the clear long-term trend of tomato yields and produce estimation errors, Root Mean Squared Errors (RMSEs), as low as 0.4kg/m^2. In addition, Tang also determined the strong linear correlation between the cumulative light intensity and the total yield discovered by McAvoy (1989). A topological case-based modelling system was developed by Hoshi (2000) to predict the daily harvest of cherry tomatoes in Japan. The system makes use of previous daily harvest, total man-hours of daily working, daily solar radiation and daily air temperature as the input variables. The evaluation results showed that the main factors influencing the daily harvest were the past yield and the total working man-hours, rather than the environmental factors.

Attempts have been made to use IS techniques, such as NNs and fuzzy logic, to determine plant development and production modelling for decades (Kaul et al., 2005; Kehagias et al., 1998; Pandey et al., 2010; Prasad et al., 2006; Simpson, 1994; Stoikos, 1995). In the recent work of Fitz-Rodriguez and Giacomelli (2009), time-delay neural networks (TDNNs) were implemented to predict the seasonal and weekly yield variation taking into account over 10 factors, including plant observations (stem diameter, number of leaves, etc.), aerial environmental conditions (air temperature, solar radiation, carbon dioxide, etc.) and root zone environment conditions (irrigation, hydrogen ion concentration, etc.). The results showed that the TDNN model can accurately estimate the weekly fluctuations of fruit-related parameters with determination coefficients (R^2) up to 0.92.

4.3 Datasets

The dataset used in this chapter was collected during a series of experiments carried out at Warwick Horticultural Research International (WHRI) over the period 1999 and 2007 to study the influence of various environmental conditions on the growth and yields of greenhouse tomatoes. In the experiments, the weekly tomato yields in a greenhouse and the daily internal environmental measurements of the greenhouse, comprising of temperature (day time average, night time average and 24hr average), solar radiation (day average), CO_2 concentration (day average) and vapour pressure deficit (VPD, day time average, night time average and 24 hr average), were collected. The yield data used in the development of the prediction system was recorded on a weekly basis since mid or late March (around the 12th calendar week) when the first fruits were picked. The harvest generally lasts about 30 weeks (ends around the 42nd calendar week). During the harvest season, the tomato yields are measured in three ways, kg per m^2, number of fruits per m^2 and average fruit weight.

In all the experiments, the greenhouse was divided into four compartments, namely B8, B9, B10 and B11. The compartments were subject to different environment regimes, such as temperature difference, lighting difference, etc. The environment conditions in the compartments were monitored and stored separately. The tomato plants placed in each compartment might be under different treatments, e.g. pruning a truss to five fruits, removing leaves around a truss, etc. The complete cultivation procedures and related treatments are described by Adams (Adams et al., 2001b; Adams & Valdes, 2002). In this work, only the yield data collected on the control plants, which grow naturally without treatment in the greenhouse, are taken into consideration to minimise the potential influences from human operations (plant pruning).

Based on the literature, all environmental factors may influence the growth of tomato plants and thus cause the fluctuations in weekly yields. Tomato fruits become more sensitive to temperature as they approach maturity (Adams and Valdes, 2002; Adams et al., 2001a); solar radiation and CO_2 are the sources of photosynthesis and thus the key factors in tomato growth and production (O'Kane, 1973; Willits and Peet, 1998); high VPD enhances the variability of tomato fruit weight (Leonardi et al., 2000); past yield was identified as a main factor influencing tomato harvest (Hoshi et al., 2000).

In the dataset, the environmental factors were stored daily and the tomato yields were stored weekly. In this work, instead of using the daily environmental measurements, the weekly averages are calculated based on the calendar weeks to be used as the inputs due to the fact that tomato growth and the environmental influence are long-term processes. The tomato fruit growing cycle (from anthesis to maturity) is about 60 days (Adams, 2002; Adams et al., 2001a), depending on their genetic species and environmental condition and the earliest harvest season started at the 11th calendar week for the experiments. Based on these, an assumption is made that the yield of fruits might be related to the environmental factors up to 10 weeks in the past. This assumption (10 weeks) should be long enough to cover the development cycle of greenhouse tomatoes under various conditions. In addition, the yields in previous weeks might also be used as indicators for the yield in the upcoming week. For those non-harvest weeks (1st week – 10th week), the weekly yields are padded with zeros as the inputs to a regression model cannot be blank and zero is a fair value to replace any non-recorded/missing value. Among all these variables, the environmental data, including temperature, radiation, VPD and CO_2 density, are considered as the independent variables and the yield is the dependent variable. The relationship with the yield in a certain week could be expressed using the following equation:

$$Y_n=F(Y_{n-1},\ldots,Y_{n-10}, R_{n-1},\ldots,R_{n-10}, T_{n-1}, \ldots,T_{n-10}, C_{n-1}, \ldots,C_{n-10}, V_{n-1}, \ldots,V_{n-10}) \quad (0.1)$$

where *Y* represents the yield, *R* represents radiation, *T* represents temperature, *C* represents CO_2 density, *V* represents VPD and the subscripts *n*, *n-1*, …, *n-10* indicate the corresponding week, i.e. *n*th week, *(n-1)*th week, etc.

Table 4.1: Basic statistics of the variables

	Unit	Min.	Max.	Mean	Standard Deviation
Yield	Kg/m^2	0.03	3.61	1.61	0.64
Temperature	^oC	16.50	24.44	19.27	1.54
Radiation	$MJ/m^2/day$	1.05	17.41	7.71	3.85
VPD	kPa	0.33	1.03	0.61	0.14
CO_2 density	p.p.m.	385.74	1152.00	691.86	161.74

Table 4.1 lists the basic statistics of the dataset after taking the weekly averages of the environmental data. It is evident that these variables are in distinct ranges, especially the measurements of the CO_2 density.

4.4 Data Pre-processing

All these variables (CO_2 density, radiation, temperature, VPD and yield) will be scaled individually using min-max scaling (equation 3.1) to transform them into the range [0, 1]. After rescaling, the variables are restructured into a standard format (50 inputs and 1 output) based on equation 4.1. For the prediction of yield in a certain week, 10 past measurements from each of the variables (radiation, temperature, CO_2, VPD and yield) are required as the input variables for the NNs. After the restructuring, the size of the input variables increased to 50 representing 10 past measurements from each of the variables. Figure 4.1 illustrates the restructuring process.

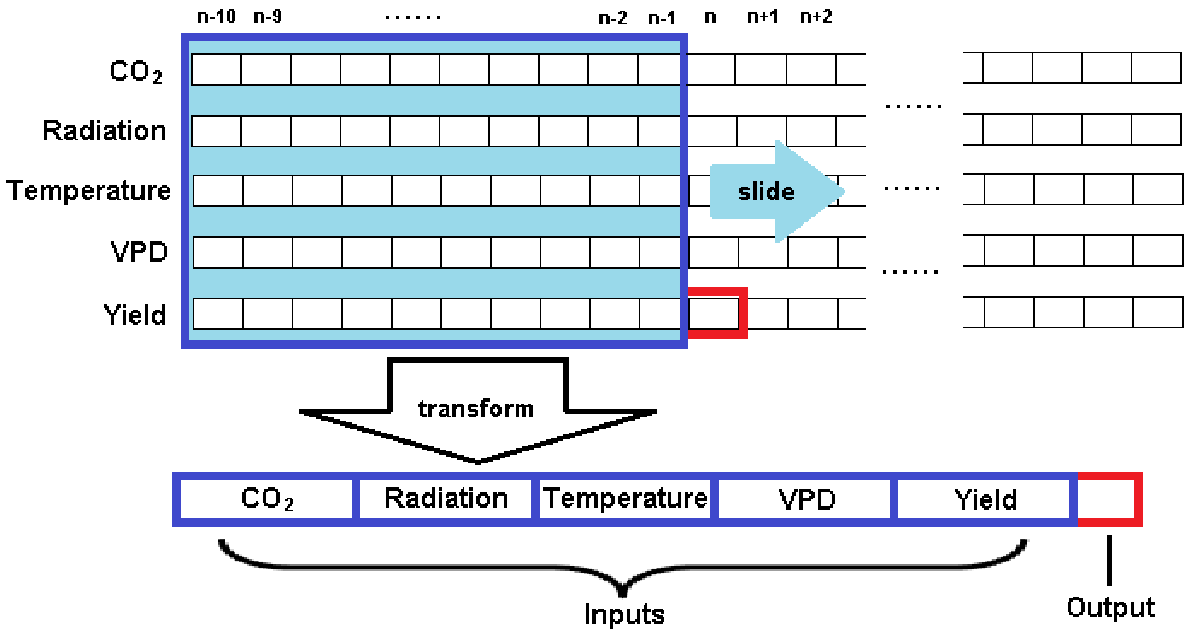

Figure 4.1: Restructuring the variables

As introduced in the previous section, 3.2.2.3.2, five-fold cross-validation will be applied to test the performance of the NNs. Hence the restructured dataset is divided into five groups of roughly equal size, so that 60% of the samples are assigned to the training set, 20% to the validation set and 20% to the test set. Table 4.2 lists the distribution of the samples.

Table 4.2: Sample distribution in the five-fold cross-validation

	Set A	**Set B**	**Set C**	**Set D**	**Set E**	**Total**
Count	178	176	184	171	175	884

4.5 Variable Selection using Sensitive Genetic Neural Optimisation (SGNO)

The reason for using SGNO in this application is to determine the influence/importance of each input variable in the production of the output. Then, a small group of variables of high influence can be used as the representatives for all the variables. As introduced in the previous section 3.2, SGNO consists of three key components, which are the Genetic Algorithm (GA) module, Neural Network (NN) module and Sensitivity Analysis (SA) module. The GA module determines the general structure of the optimisation algorithm. The NN module functions as part of the GA module and evaluates the performance of the potential solutions generated in the GA module. The SA module provides a refined analysis of a group of potential solutions of high performance, i.e. low prediction errors, to determine the global influence of each variable.

4.5.1 Neural Network module

NNs are not only used as part of the SGNO algorithm to evaluate the chromosomes in the GA module, they are also used as the modelling tool to re-evaluate the input variables selected by the SGNO. When a NN is used as the NN module in SGNO, the number of hidden layers is fixed to be 1, as NNs of a single hidden layer are known to be good universal approximators and training NNs of single hidden layer is more computationally efficient than NNs with multiple hidden layers (Csaji, 2001; Heaton, 2005; Hornik et al., 1989). The number of hidden neurons in NNs is estimated dynamically as half of the sum of inputs and outputs (Equation 3.5) instead of using a fixed number of hidden neurons. In addition, random initial weights are used in the NNs. The selection of activation functions is generally application oriented and an activation function may only work well for a specific type of application. In the case of this tomato yield prediction application, the well-known tangent sigmoid function, aka hyperbolic sigmoid function, (Equation 2.5) is used in the hidden neurons and the output neuron. The tangent-sigmoid function is continuous non-linear and scales the input in the range $[-\infty, \infty]$ to $[-1, 1]$.

4.5.2 Genetic Algorithm module

The GA module constructs the general structure of the SGNO algorithm as illustrated in Figure 3.1 (see Section 3.1). In this tomato yield prediction application, the initial GA population consists of a group of randomly generated 50-bit binary chromosomes (potential solutions). The size of the initial population is estimated using the rule of thumb, equation 3.3 (Cox, 2005). As there are 50 input variables, the total search space would have $2^{50}-1 \approx 1x10^{15}$ states. Hence the size of the initial population is estimated to be 250.

Evaluation of the chromosomes' fitness is performed using the NN module with five-fold cross-validation. In each generation, an elite group of chromosomes with the highest fitness in the population are selected and passed to the new generation without going through the crossover and mutation operations to ensure some of the good features are preserved in the new generation. In this application, 12 elite chromosomes, which are approximately 5% of the population, are selected in each generation. The chromosome mutation rate is set to be 5% and the GA stopping criterion is the evaluation of 50 generations.

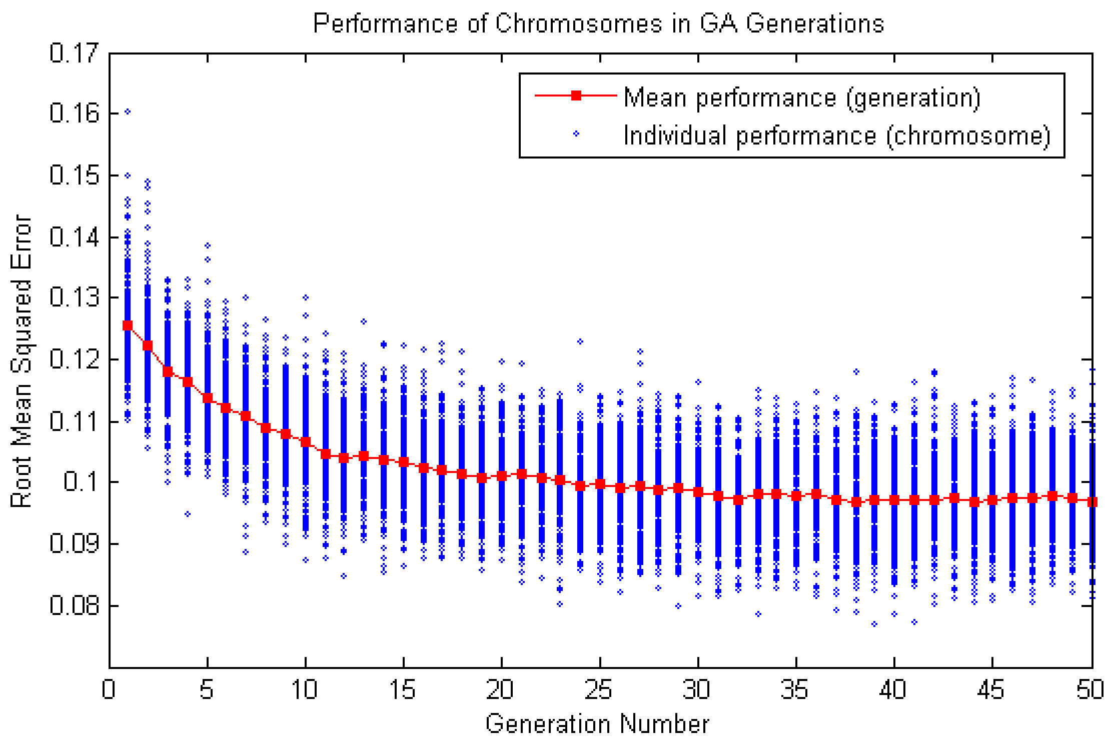

Figure 4.2: Performance of chromosomes in GA generations.

Figure 4.2 illustrates the performance of the chromosomes in each generation. As the GA evolves, it can be seen in the figure that the mean error of chromosomes decreases rapidly in the first 10 generations and then slows down in the following 20 generations. The mean error of a generation tends to converge from the 30th generation. For the individual chromosomes, the error of a chromosome in later generations is not guaranteed to be lower than the ones in former generations.

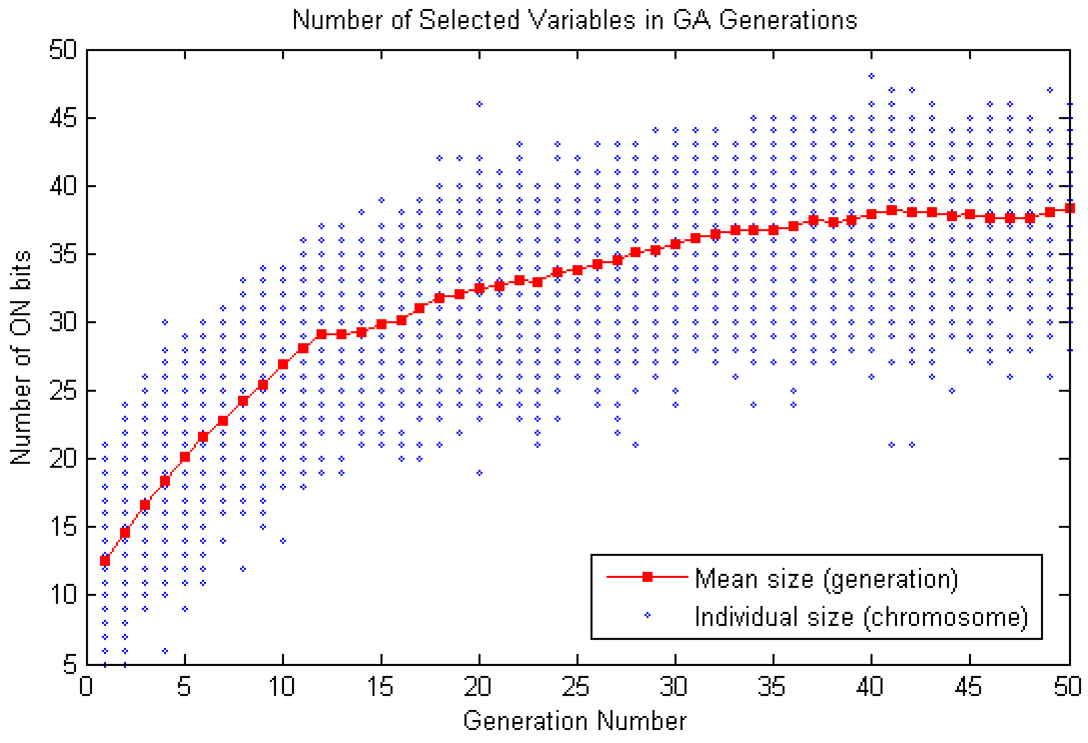

Figure 4.3: Number of selected variables (ON bits) in each generation.

Figure 4.3 illustrates the changes in the sizes of chromosomes, which are the numbers of selected variables (ON bits) in individual chromosomes, as the GA evolves. It clearly shows that the mean chromosome size increases from approximately 12.5 in the first generation to 39 in the 50th generation. The mean chromosome size grows rapidly in the first 12 generations then increases gradually at a reduced rate until approximately the 40th generation, beyond which it tends to converge.

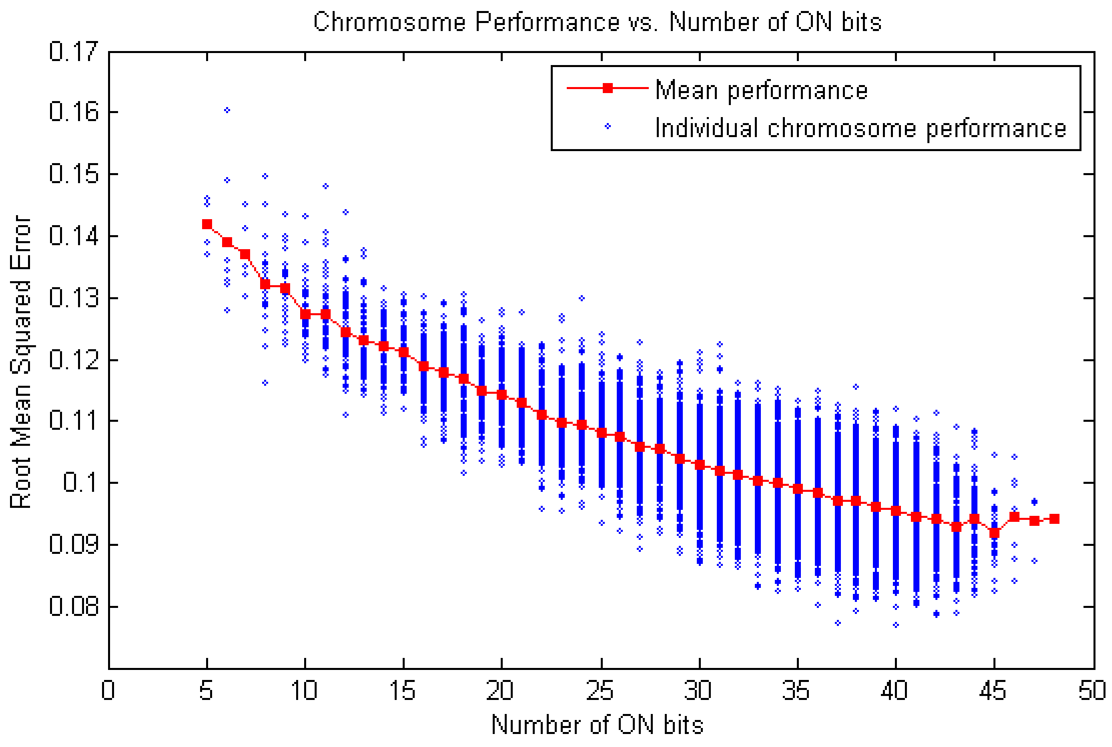

Figure 4.4: Chromosome Evaluation Error vs. Number of Selected Variables.

Figure 4.4 illustrates the relationship between the performance error (RMSE) and the number of ON bits in the chromosomes. A near-linear relationship is observable between the mean performance error and the number of ON bits. The performance error decreases as the number of ON bits increases. The general negative correlation trend suggests that almost all the input variables are contributing to the output. However, it is not necessarily the case that more ON bits would produce lower error. Chromosomes of the same number of ON bits produce different errors covering a broad range of RMSEs. Possible factors that cause the variation in performance include:

- The various combinations of the ON bits in chromosomes. Different variables may have different influences in the NN models; some variables may have strong contributions towards the model output and some may have minor or even negative contributions, i.e. instead of improving the NN model's performance, they reduce its performance when added to the NN model.
- The performance of the NN module that evaluates the chromosome. Every chromosome is evaluated by a number of NNs with randomised initial weights, which usually lead to minor differences in the outputs produced. Hence, even identical chromosomes will have different fitness values generated by the NN module.

4.5.3 Sensitivity Analysis module

In the SA module, a quarter of the chromosomes evaluated in each generation of the GA module that produces the lowest errors are selected to carry out the refined SA analysis on each of the input variables. In this application, the GA population size is 250 and they have been evaluated for 50 generations. Hence, 62 (250/4≈62) chromosomes will be selected from each generation and the total number of chromosomes to carry out the SA analysis is 3100 (62x50=3100). Figure 4.5 illustrates the size (number of selected variables) of these chromosomes selected for the SA analysis. It shows clearly that most of the selected chromosomes have a large number of ON bits (over 30) due to the fact that the number of ON bits in the chromosomes increases as the GA evolves (see

Figure 4.3) and the chromosomes with larger numbers of ON bits are more likely to produce better performance (see Figure 4.4).

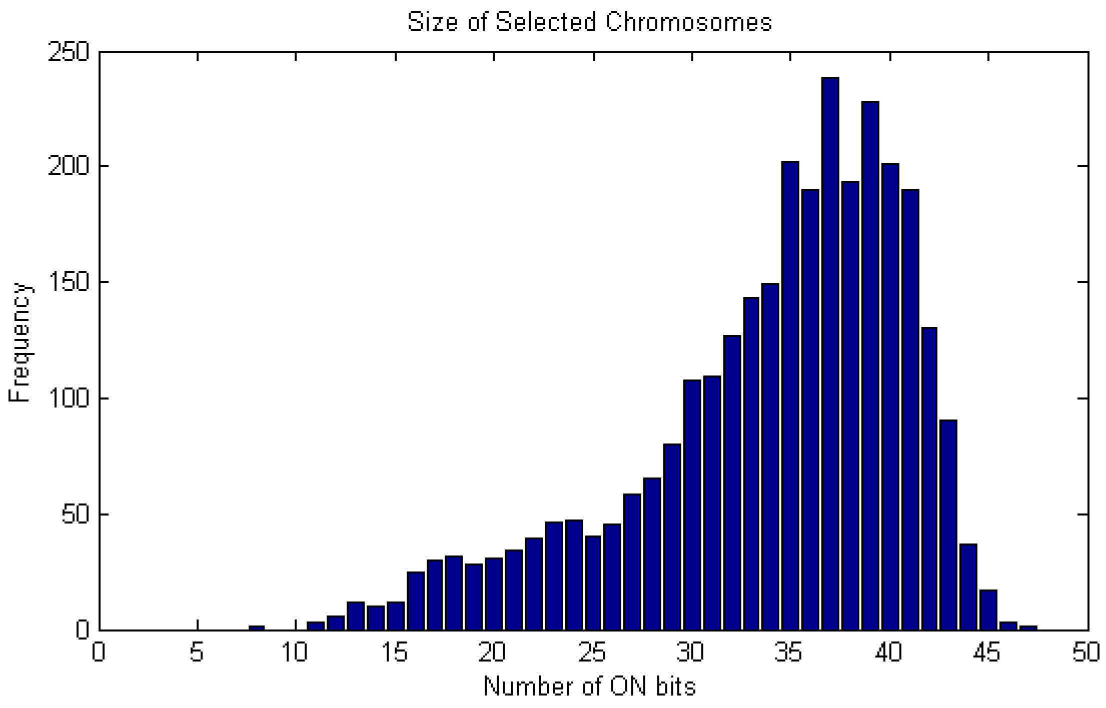

Figure 4.5: Size of the chromosomes selected for SA analysis

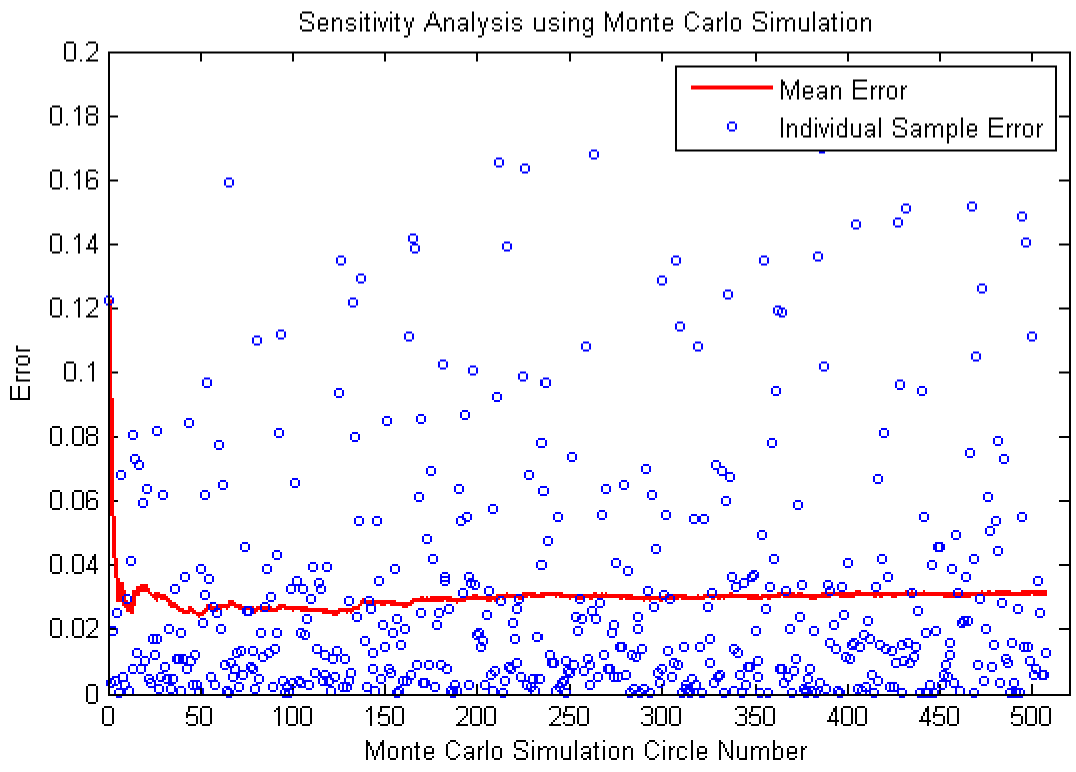

Figure 4.6: Monte Carlo simulation

The SA analysis used in SGNO is based on the Monte Carlo simulation (MCS). The sensitivity score of a certain variable is defined as the mean difference if it is replaced by an arbitrary number. Figure 4.6 illustrates the MCS of a variable in a randomly picked chromosome. In this figure, the mean error plot starts converging after taking approximately 160 random samples. The converged mean error is considered as the sensitivity score of the variable in that randomly selected chromosome. Although the Monte Carlo simulation converges after taking about 160 samples, more random samples were actually taken as shown in Figure 4.6. The reason for taking more samples is due to the fact that the training and evaluation (simulation) of NNs in Matlab® is operationally expensive. Hence, it is preferable to perform training and evaluation on a large chunk of data rather than repetitively performing the operations on the same amount data in smaller blocks.

After computing the variables' sensitivity scores in all the selected chromosomes, the global sensitivity score of each variable can be derived by calculating the mean value. Figure 4.7 illustrates the global sensitivity scores for all variables.

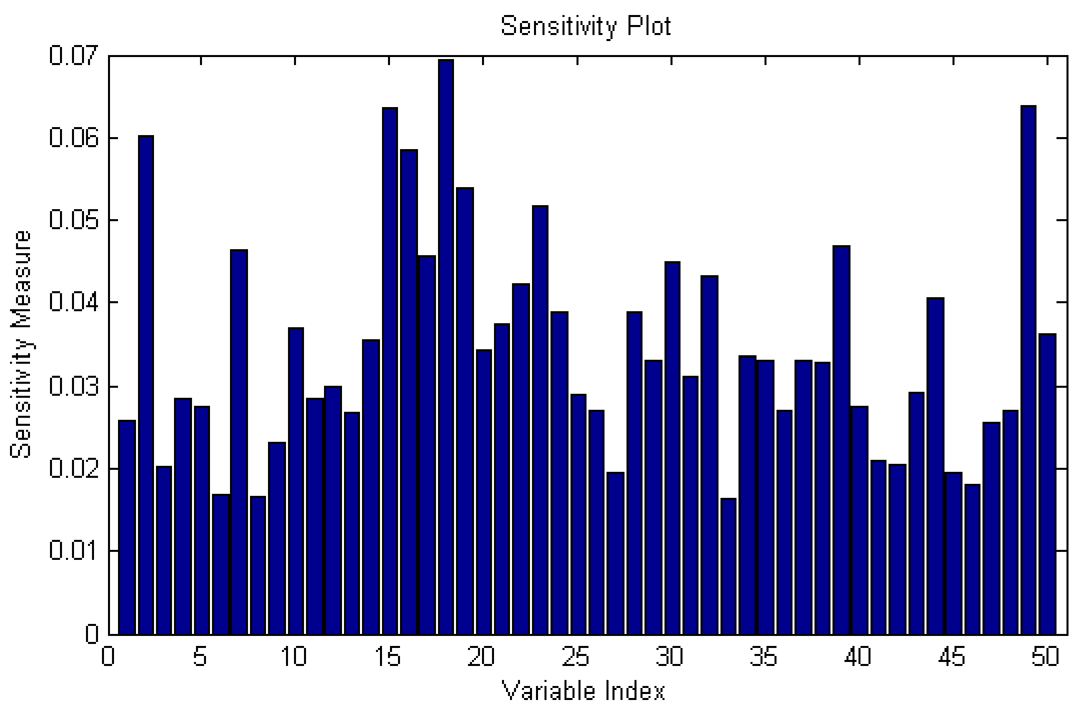

Figure 4.7: Global sensitivity scores for all variables

As the sensitivity score of an input variable reflects its influence or contribution to the output of the model, a variable of higher sensitivity score is considered more important/influential than a variable of lower sensitivity score. By rearranging the input variables according to their sensitivity scores, an importance rank table can be constructed, which is listed below. The list is the importance rank table containing the variable indices in GA chromosomes arranged in the descending order of sensitivity scores from the highest to the lowest.

[18, 49, 15, 2, 16, 19, 23, 39, 7, 17, 30, 32, 22, 44, 28, 24, 21, 10, 50, 14, 20, 34, 29, 35, 37, 38, 31, 12, 43, 25, 11, 4, 40, 5, 26, 48, 36, 13, 1, 47, 9, 41, 42, 3, 45, 27, 46, 6, 8, 33]

The ordered variable indices in the importance rank table represent the positions in the chromosome. Table 4.3 shows the indices in the chromosomes and their corresponding information.

Table 4.3: GA Chromosome explained

CO_2									
Index in chromosome									
1	2	3	4	5	6	7	8	9	10
Represent data collected in week									
n-10	n-9	n-8	n-7	n-6	n-5	n-4	n-3	n-2	n-1

Radiation									
Index in chromosome									
11	12	13	14	15	16	17	18	19	20
Represent data collected in week									
n-10	n-9	n-8	n-7	n-6	n-5	n-4	n-3	n-2	n-1

Temperature									
Index in chromosome									
21	22	23	24	25	26	27	28	29	30
Represent data collected in week									
n-10	n-9	n-8	n-7	n-6	n-5	n-4	n-3	n-2	n-1

VPD									
Index in chromosome									
31	32	33	34	35	36	37	38	39	40
Represent data collected in week									
n-10	n-9	n-8	n-7	n-6	n-5	n-4	n-3	n-2	n-1

Yield									
Index in chromosome									
41	42	43	44	45	46	47	48	49	50
Represent data collected in week									
n-10	n-9	n-8	n-7	n-6	n-5	n-4	n-3	n-2	n-1

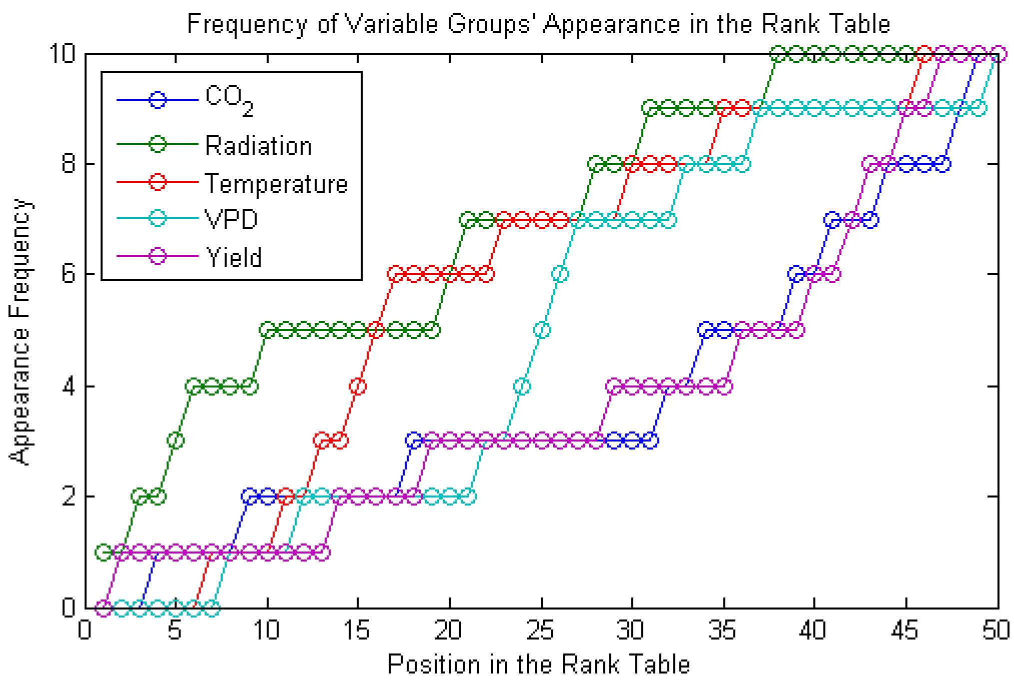

Figure 4.8: Appearance of Variable Groups in the Rank Table

Based on the importance rank table, the appearance of a certain group of variables (CO_2, radiation, etc.) can be calculated. Figure 4.8 illustrates the accumulated appearance frequencies of the five variable groups in the rank table. In the figure, the appearance of radiation is dominant when a small number of variables, say 15, is selected to represent the original data. The appearance of temperature quickly rises following the radiation. When 20 variables are selected to represent the original data, temperature and radiation become the most important variable groups. The appearances of the CO_2 and yield variables gradually increase indicating that these two groups are generally less important than the other variable groups. To sum up, the radiation is no doubt the most influential variable group as it always has the highest appearance frequencies. The temperature group is of similar level of importance as the radiation group. The VPD group is of medium influence in general, while the CO_2 and yield are less influential on average.

4.6 System Remodelling

As mentioned in the previous section 3.2.3, NNs are not only used as the fitness functions in the GA module, but are also used to re-evaluate the performance of variables selected in the SGNO. By selecting various numbers of variables from the beginning of the importance rank table, it is presumed that a greater number of variables would produce better results. Table 4.4 lists the re-evaluation results, including RMSE and coefficient of determination (R^2). The 'Size' rows indicate the number of variables selected from the rank table. Figure 4.9 illustrates the results of taking 5, 10 and 15 inputs from the importance rank table and figure 4.10 illustrates the re-evaluation results (RMSE) as compared to the chromosomes evaluated in the SGNO's GA module.

Table 4.4: SGNO remodelling results

Size	1	2	3	4	5	6	7	8	9
RMSE	0.1392	0.1377	0.1374	0.1333	0.1282	0.1261	0.1217	0.1198	0.1194
R^2	0.3678	0.3815	0.3839	0.4204	0.4643	0.4815	0.5173	0.5317	0.535

Size	10	11	12	13	14	15	16	17	18
RMSE	0.1193	0.1169	0.1144	0.1136	0.1131	0.1089	0.1088	0.1063	0.1069
R^2	0.5357	0.5546	0.5733	0.5792	0.583	0.6134	0.6142	0.6312	0.627

Size	19	20	21	22	23	24	25	26	27
RMSE	0.1019	0.103	0.1061	0.1035	0.1027	0.1015	0.1015	0.1001	0.101
R^2	0.6616	0.6541	0.6326	0.6505	0.6559	0.6637	0.6643	0.6731	0.6675

Size	28	29	30	31	32	33	34	35	36
RMSE	0.101	0.098	0.0968	0.0971	0.0986	0.0954	0.0944	0.0945	0.0966
R^2	0.667	0.6868	0.6946	0.6926	0.6832	0.703	0.7092	0.7088	0.6958

Size	37	38	39	40	41	42	43	44	45
RMSE	0.093	0.0935	0.0938	0.0948	0.0926	0.0917	0.0889	0.0924	0.0911
R^2	0.718	0.7147	0.7132	0.7068	0.7203	0.7256	0.7423	0.7215	0.7293

Size	46	47	48	49	50
RMSE	0.0917	0.0898	0.0879	0.0906	0.0889
R^2	0.726	0.7373	0.7483	0.7324	0.7423

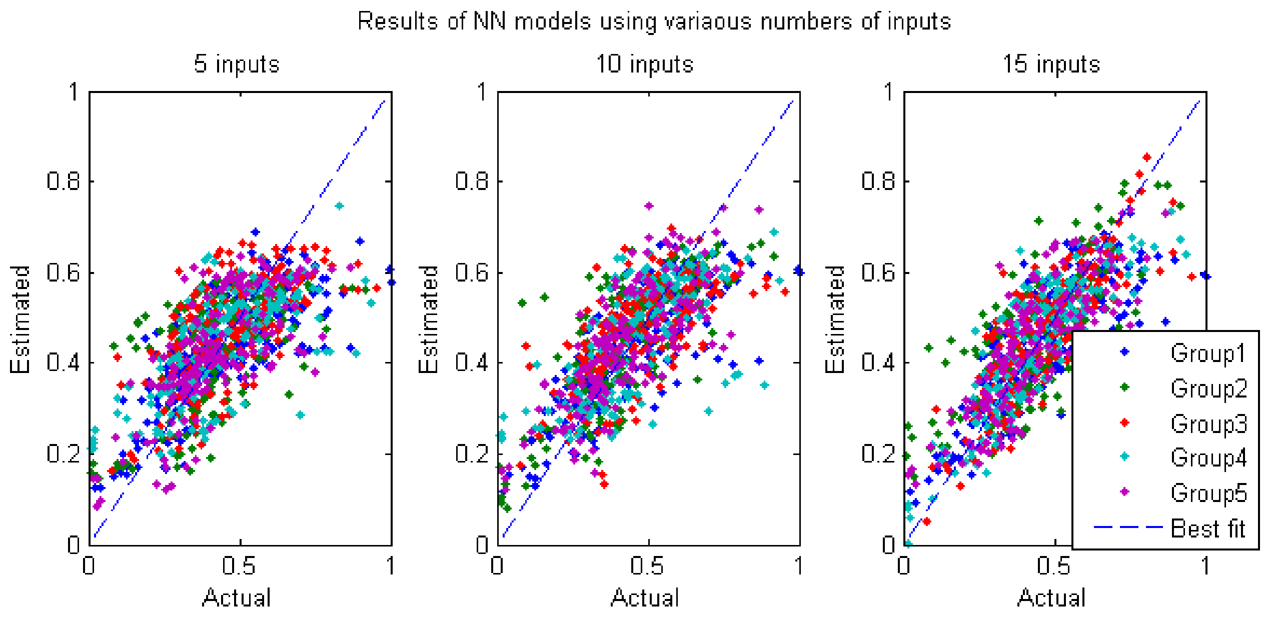

Figure 4.9: Remodelling results of various numbers of inputs

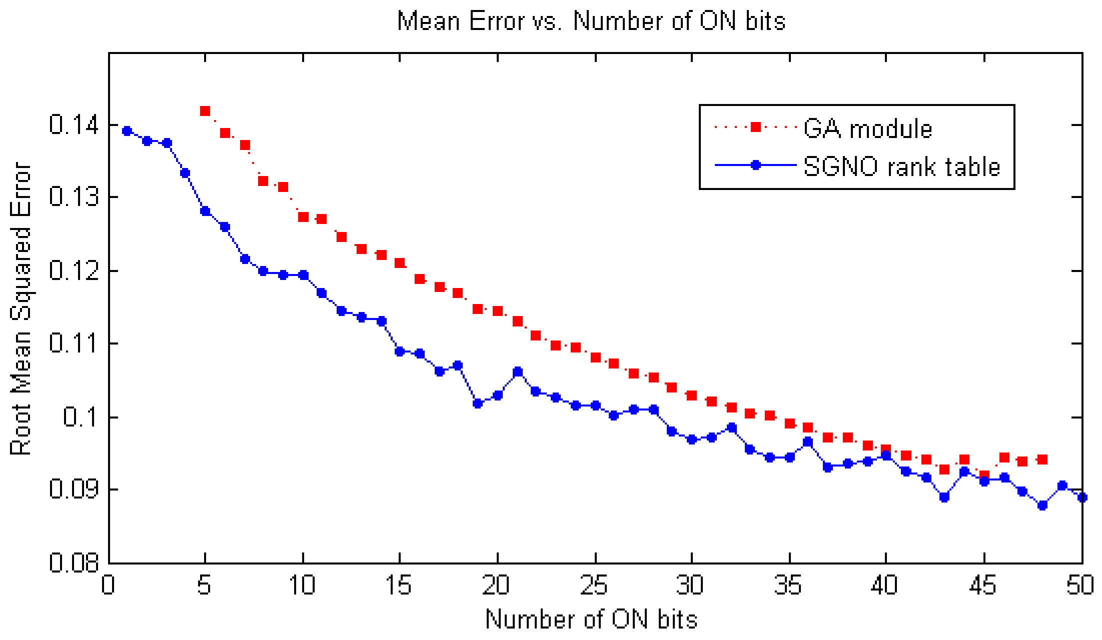

Figure 4.10: Performance of variables in SGNO rank table against chromosomes evaluated in the GA module

As illustrated in Figure 4.10, the variables selected from the SGNO rank table always perform better than the average performance of the same number of variables selected by chromosomes in the GA module. The advantage of using the variables in the rank table gradually reduces as the number of selected variables increases. The plot of the SGNO's performance shows a decreasing trend. However, there are certain points on the plot that do not follow the trend nicely. This is due to the fact that NNs are non-deterministic models; different initial weights in hidden neurons may lead to different generalised states after training, and thus generating different outputs. On the other hand, some of the variables or the combinations of some variables may have negative contributions to the NN models and thus reduce the performance of the NN models when they are used as the inputs.

The architectures of the NNs used so far are not optimal as the architectures of the NN models are estimated using rule of thumbs, which suggests that a single layer of hidden neurons is sufficient and the number of hidden neurons is determined by halving the sum of the numbers of inputs and outputs. Hence the regression results produced using the proposed variable combinations may not be optimal. To explore the best performance achievable using the variable combinations in the importance rank table, NNs of 'optimal' architectures taking various numbers of input variables (5-variable, 10-variable, 15-variable and 20-variable) are discovered. The 'forward' selection procedure, which was explained in section 3.2.3, is implemented to find the best architecture, in which the number of hidden neurons gradually increases until a certain condition is reached. The following restrictions are implemented to limit the search domain of the 'forward' selection procedure (Heaton, 2005):

- The number of hidden neurons should be no more than twice the input layer size.

$$\boldsymbol{N_{hidden} \leq 2N_{inputs}} \tag{4.1}$$

- There are no more than two hidden layers.

The 'optimal' architecture of a certain NN is discovered using the following steps:

1. A single hidden layer NN of only 1 hidden neuron is first evaluated (The NN is trained using the dataset and the training error is obtained to represent the performance of the NN).
2. An extra hidden neuron is added to the NN and the NN is evaluated.
3. Step 2 is repeated until the number of hidden neurons is equal to twice the number of input variables.
4. An extra hidden layer with one hidden neuron is added to the NN, the first hidden layer is reset to 1 hidden neuron and re-evaluation is performed.
5. An extra hidden neuron is added to the first hidden layer and re-evaluation is performed.
6. Step 5 is repeated until the total number of hidden neurons is equal to twice the input variables.
7. An extra hidden neuron is added to the second hidden layer, the first hidden layer is reset to 1 hidden neuron and re-evaluation is performed.
8. Steps 5 to 7 are repeated until the number of hidden neurons in the second hidden layer is equal to twice the input variables.

9. The evaluation results at each stage are stored in a matrix. The row index of the matrix represents the number of hidden neurons in the first hidden layer, i.e. the first row indicates the presence of 1 hidden neuron, the second row indicates 2 hidden neurons, etc. The column index represents the number of hidden neurons in the second hidden layer minus 1, i.e. the first column indicates that the second hidden layer does not exist, the second column indicates the presence of 1 hidden neuron in the second hidden layer, etc.
10. The best performance can be identified by locating the lowest evaluation result in the matrix and the location of the result indicates the 'optimal' architecture.

Figure 4.11 illustrates the matrix containing the evaluation results for the NN with 5 input variables selected from the importance rank table. The x-axis indicates the number of hidden neurons in the second hidden layer; the y-axis indicates the number of hidden neurons in the first hidden layer; the colours of the pixels indicate the evaluation results.

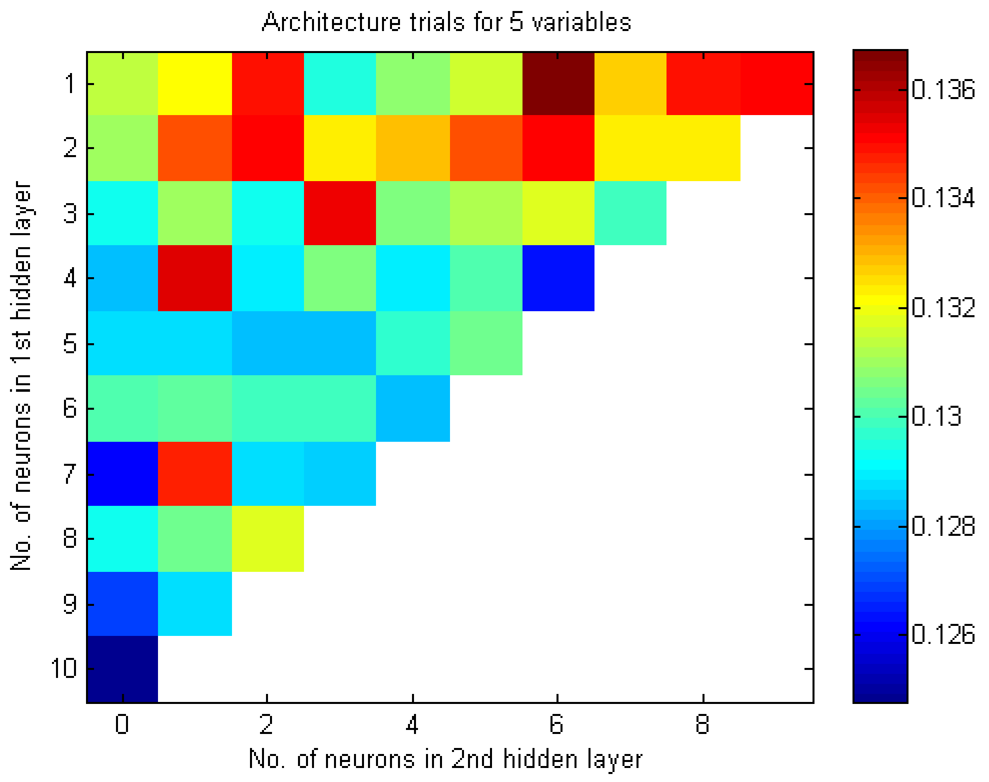

Figure 4.11: Architecture trials for NNs of 5 input variables

As can be seen in Figure 4.11, the lowest performance occurred at the point [0, 10] representing the NN architecture of 10 neurons in the first hidden layer and no second layer. In this 5-variable situation, a single hidden layer performs well in general and the extra hidden layer does not contribute positively to the NN under the predefined restrictions. Table 4.5 shows the 'optimal' architectures discovered for the various key points and their corresponding performance. Based on the discovery listed in Table 4.5, all the 'optimal' architectures have a single hidden layer with a large number of hidden neurons as compared to the number of input and output neurons.

Table 4.5: Discovery of 'optimal' architectures and their performances

Number of input variables	'Optimal' architecture [1st layer, 2nd layer]	Performance (RMSE)
5	[10, 0]	0.1247
10	[18, 0]	0.1103
15	[30, 0]	0.0977
20	[33, 0]	0.0917

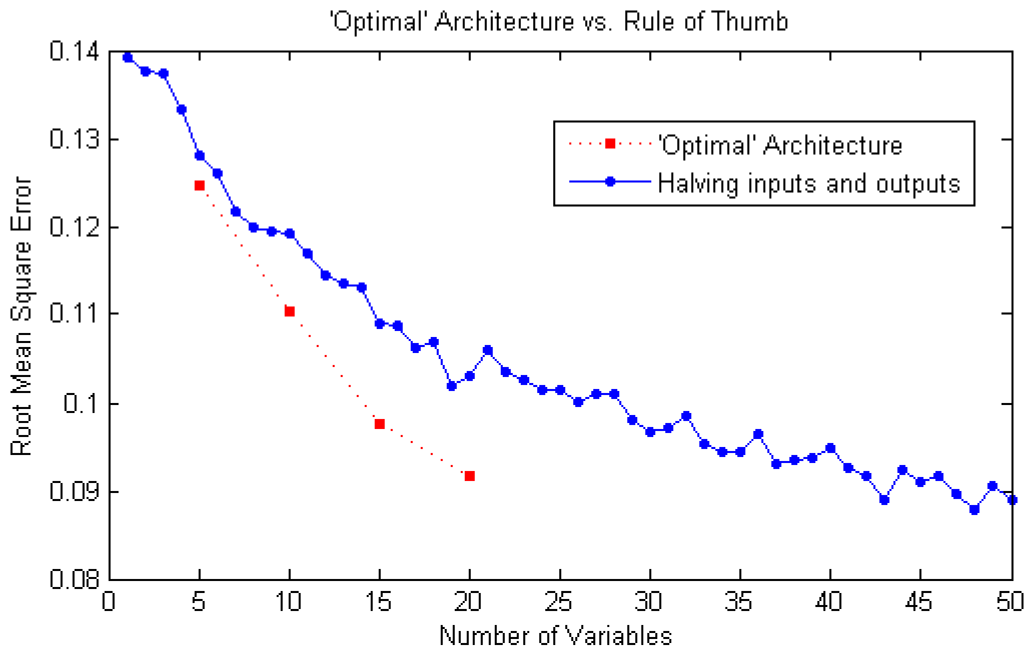

Figure 4.12: Performance of NNs of 'optimal' and of estimated architectures

Figure 4.12 illustrates the performance of the NNs of 'optimal' architectures those NNs with estimated architectures. As illustrated in the figure, the NNs of 'optimal' architectures perform significantly better than the architectures estimated using the rule of thumb.

4.7 Symbolic Regression using Genetic Programming

NNs are generally known as black-box models as the physical parameters of the NNs are generally not understandable. In SGNO, Genetic Programming (GP) with Multiple Branches Encoding (MBE) is implemented to discover simple symbolic relationships (symbolic regression) between the selected input variables and the outputs. The size (depth) of the tree structured expressions and the number of number of tree branches in the GP chromosomes are restricted to a certain number, which can only be changed when necessary to control (limit) the expansion of the expressions. In this application, seven commonly used mathematical operators are selected as the source of expressions. These operators are listed in Table 4.6.

Table 4.6: Mathematical operators in GP

Operator Name	Expression	Arity
Plus	+	2
Minus	-	2
Times	*	2
Divide	/	2
Power	POW	1
Logarithm	LOG	1
Square root	SQRT	1

For the case of 5 input variables, the population size is estimated at 175 (5x7x5=175) using the rules of thumb (Equation 3.10), the initial maximum number of expression tree branches in a chromosome is 5 and the initial maximum tree depth is 3. In addition, 8 elite chromosomes, which are about 5%

of the population, in each GP generation are passed to the new generation without modification.

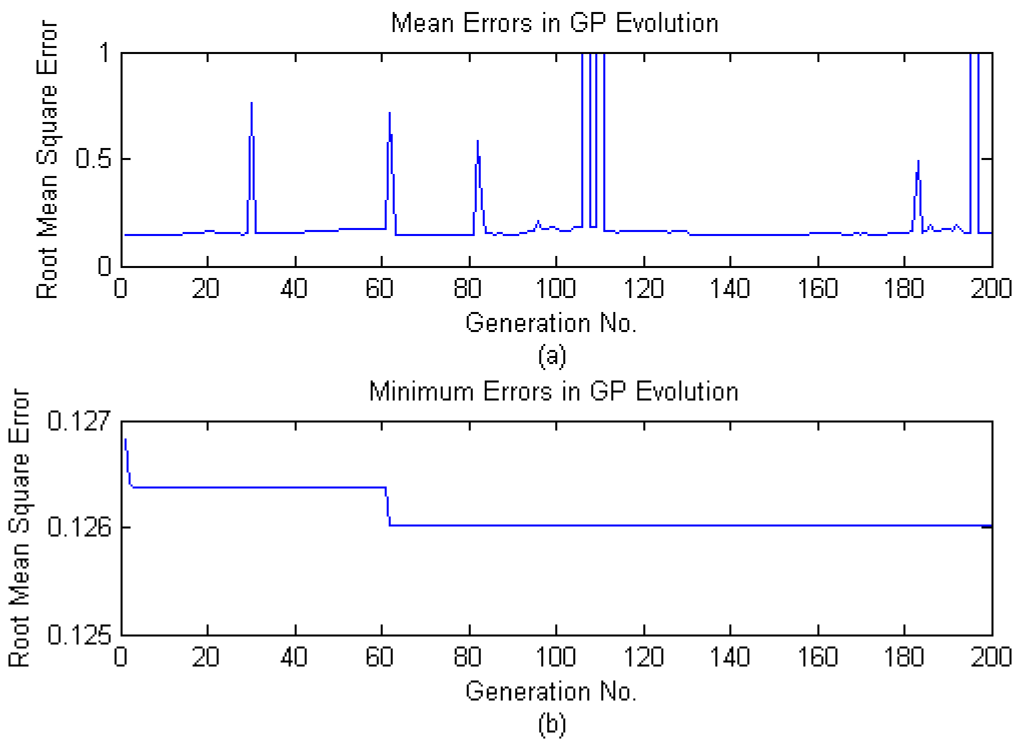

Figure 4.13: Evolution of GP for 5 variables. (a) mean chromosome error in each generation; (b) minimum chromosome error in each generation

Figure 4.13 illustrates the evaluation of the GP process in the case of 5 input variables from the rank table. It shows that the chromosome of the lowest error, 0.1260, is produced at the 61st generation and the chromosome is expressed by the equation 4.3. This MBE-GP chromosome produces comparable performance to the 'optimal' NNs with the same input variables, which generates a mean error of 0.1247 as listed in table 4.5. Comparing with the NNs of estimated architectures, which produce a mean error of 0.1282 (see table 4.4), this symbolic expression is of minor advantage.

$$\boldsymbol{y = A * LOG(x_1) + B * x_4 * x_3 + C * LOG({x_5}^{0.2}) + D * {x_2}^4 + E} \qquad (4.2)$$

where *A, B, C, D* and *E* are the coefficients to be determined during the evaluation of the chromosome; x_1, x_2, x_3, x_4 and x_5 are the corresponding variables in the rank table.

The following symbolic expression, equation 4.4, is discovered if 10 input variables are selected. This GP chromosome produces an average error as low as 0.1214. Comparing with the 'optimal' NNs with the same input variables, which produce an average error of 0.1103 and the NNs of estimated architectures that generate an average error of 0.1193, this symbolic expression is competitive to the 'non-optimal' NNs but not as the 'optimal' NNs.

$$\boldsymbol{y} = \boldsymbol{A} * \left(\boldsymbol{x_9} + \boldsymbol{x_5} + \sqrt{\boldsymbol{x_6}} + \boldsymbol{x_{10}}\right) + \boldsymbol{B} * \left(\sqrt{(\boldsymbol{x_2 x_7})^4} + \boldsymbol{x_9}^5\right) + \boldsymbol{C} \qquad (4.3)$$

4.8 Benchmarking with other Feature Selection (FS) Techniques

In this section, the performance of SGNO is benchmarked against several other dimensionality reduction techniques, including BFS, FFS, GNMM and PCA, by evaluating the variable rank tables explored by all these techniques.

4.8.1 Principal Component Analysis (PCA)

PCA is a famous mathematical procedure that transforms a set of multivariate data linearly into a new coordinate system of the same number of dimensionality. The dimensionality reduction using PCA is to replace the original data with a number of PCs, usually less than the number of variables in the original data but carrying a significant amount of variations (Dunteman, 1989). The detailed procedures of PCA are discussed previously in section 3.3.1.

In this tomato yield prediction application, PCA is used to transform the original dataset of 50 input variables into a new coordinate system. Then various numbers of PCs are selected and used as inputs to the NN models to perform tomato yield prediction. The performance of PCA is compared with SGNO by evaluating the same number of variables selected from the variable rank table constructed by SGNO.

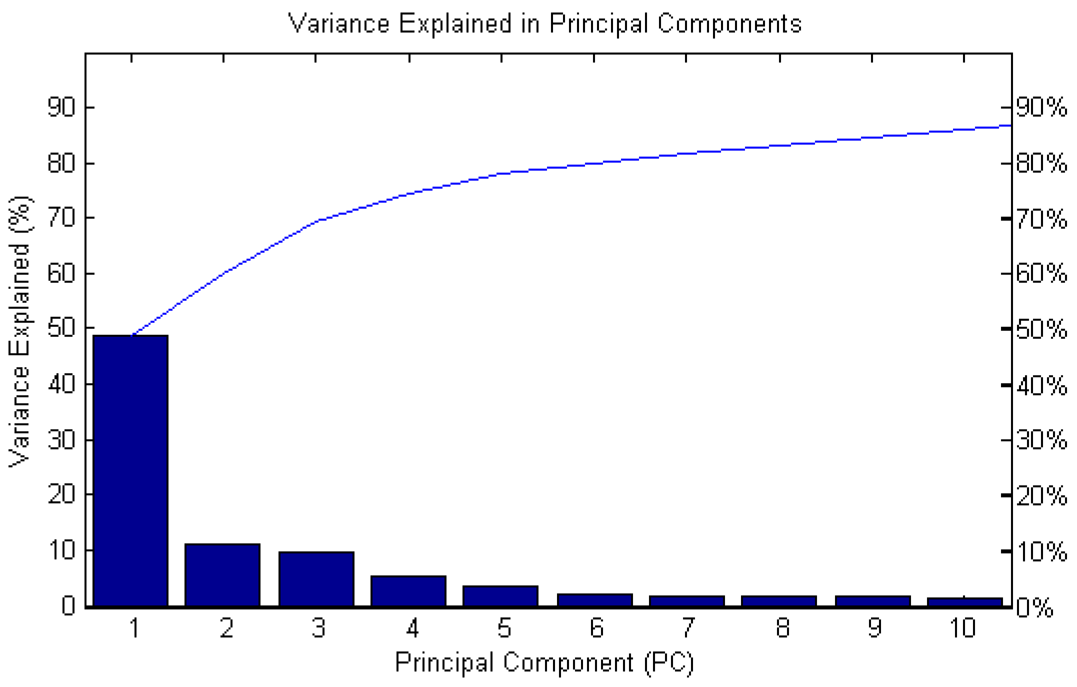

Figure 4.14: Variance accounted for or explained in PCs

Figure 4.14 illustrates the variance accounted for by each PC and there are only 10 PCs present in the figure. As illustrated in the figure, the first PC explains about 48% of the variance in the original dataset, which is significantly greater than the other PCs. The accumulated variance plot (the blue thin plot) shows that the first 3 PCs explain about 70% of the variance and the first 6 PCs explain around 80%.

Following PCA transformation, NN models are employed to predict the tomato yield using various numbers of PCs. The NN models employed contains a single hidden processing layer and the number of hidden neurons is estimated by halving the sum of the inputs and output. Table 4.7 lists the evaluation results (RMSEs) of the NN models. The 'size' rows indicate the number of PCs used as input variables.

Table 4.7: PCA evaluation results

Size	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
RMSE	0.174	0.1727	0.1656	0.165	0.1546	0.1485	0.1431	0.1436	0.1385	0.1371	0.1303	0.1288	0.1292	0.1283	0.1243	0.1234	0.1205	0.1188	0.1182	0.1181	0.1154	0.1153	0.1111	0.108	0.1072	0.1062	0.1053	0.1065	0.1062	0.1038	0.0994	0.1024	0.099	0.0985	0.0979	0.0963	0.0961	0.0948	0.0977	0.0939	0.097	0.0928	0.0934	0.0926	0.0918	0.0903	0.0917	0.0906	0.0893	0.0881

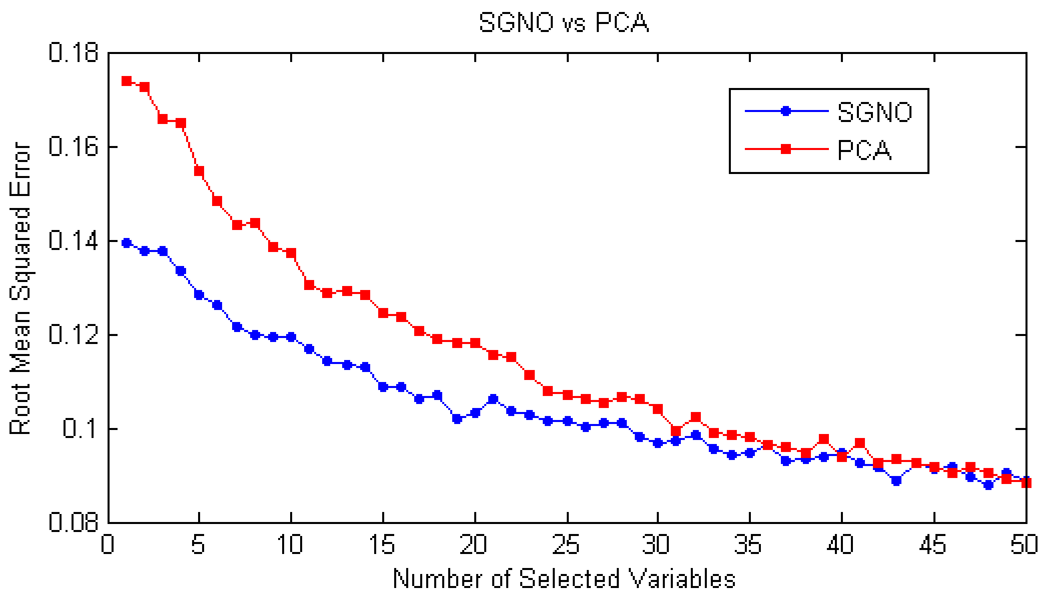

Figure 4.15: Performance of ANN models using PCA discovery against SGNO

Figure 4.15 illustrates the performance of the NN models using various numbers of PCs against the performance of the NN models taking the same number of variables selected from the SGNO's importance rank tale. The figure shows clearly that both performance plots gradually decrease as the number of input variables increases and the SGNO's rank table is always superior to the PCA. The advantage of SGNO is significant when the number of input variables is small. The largest difference between the performance of SGNO and PCA occurs when only one or two variables are used as inputs, and the difference is up to the value of 0.035. The advantage of SGNO weakens as the number of input variables increases and finally vanishes when the number of variables exceeds 35 and the performance plots start to overlap.

4.8.2 Forward Feature Selection (FFS) and Backward Feature Selection (BFS)

FFS and BFS are conventional FS techniques. The detailed explanations of these two techniques are discussed in section 3.3.2. In this application, FFS and BFS employ NN models to evaluate the variables in the variable lists and decide which variable to be added or removed based on the feedback from the NN models. The ANN models employ the same architectures as the NN module in the SGNO process, i.e. single hidden layer with the number of hidden neurons estimated by halving the sum of the inputs and outputs. The following lists present the variable preference tables constructed using BFS and FFS. The variables are in descending order of preferences in both lists.

BFS: [50, 17, 24, 43, 16, 21, 36, 49, 40, 28, 47, 30, 8, 1, 14, 35, 10, 29, 11, 46, 48, 15, 3, 20, 44, 22, 27, 13, 5, 19, 42, 45, 2, 34, 12, 26, 6, 7, 18, 38, 37, 25, 39, 31, 41, 23, 9, 4, 33, 32]

FFS: [50, 20, 48, 23, 18, 30, 43, 42, 14, 39, 41, 28, 7, 36, 38, 16, 8, 32, 47, 44, 49, 2, 9, 21, 33, 22, 6, 34, 15, 1, 29, 10, 17, 26, 5, 46, 40, 45, 12, 25, 31, 13, 11, 27, 37, 4, 24, 35, 3, 19]

Comparing with the preference list generated using SGNO, which is **[18, 49, 15, 2, 16, 19, 23, 39, 7, 17, 30, 32, 22, 44, 28, 24, 21, 10, 50, 14, 20, 34, 29, 35, 37, 38, 31, 12, 43, 25, 11, 4, 40, 5, 26, 48, 36, 13, 1, 47, 9, 41, 42, 3, 45, 27, 46, 6, 8, 33]**, these three preference lists are distinct from each other. However, both BFS and FFS identify the 50th variable as the most favourable, and both SGNO and BFS consider the 33rd variable to be unfavourable. Table 4.8 lists the evaluation results

(RMSEs) of the preference lists constructed using BFS and FFS when various numbers of input variables are selected.

Table 4.8: BFS and FFS evaluation results (RMSEs)

Size	1	2	3	4	5	6	7	8	9
BFS	0.1367	0.1313	0.1303	0.1304	0.126	0.1255	0.1246	0.1231	0.1193
FFS	0.1367	0.1311	0.13	0.1263	0.1229	0.1226	0.1213	0.1212	0.1199

Size	10	11	12	13	14	15	16	17	18
BFS	0.12	0.1189	0.1181	0.1165	0.1176	0.1152	0.1126	0.1129	0.1131
FFS	0.1197	0.1185	0.1177	0.1174	0.1162	0.1168	0.1169	0.114	0.1139

Size	19	20	21	22	23	24	25	26	27
BFS	0.1123	0.1108	0.112	0.1107	0.1085	0.108	0.1064	0.1067	0.1056
FFS	0.1134	0.1129	0.1093	0.1078	0.1075	0.1065	0.1065	0.1054	0.1039

Size	28	29	30	31	32	33	34	35	36
BFS	0.1063	0.1049	0.1038	0.1022	0.1026	0.1026	0.1017	0.0995	0.1004
FFS	0.1036	0.0993	0.1018	0.0997	0.0984	0.0995	0.0992	0.0975	0.0971

Size	37	38	39	40	41	42	43	44	45
BFS	0.1003	0.0975	0.0987	0.0949	0.0961	0.0952	0.0945	0.0913	0.0916
FFS	0.0964	0.0968	0.0934	0.094	0.0933	0.0931	0.0929	0.0935	0.0927

Size	46	47	48	49	50
BFS	0.089	0.0906	0.0926	0.0886	0.0894
FFS	0.091	0.094	0.0914	0.0904	0.0894

Figure 4.16 illustrates the difference in performance of BFS and FFS against SGNO in this task of estimating weekly tomato yields using various numbers of variables from the preference lists.

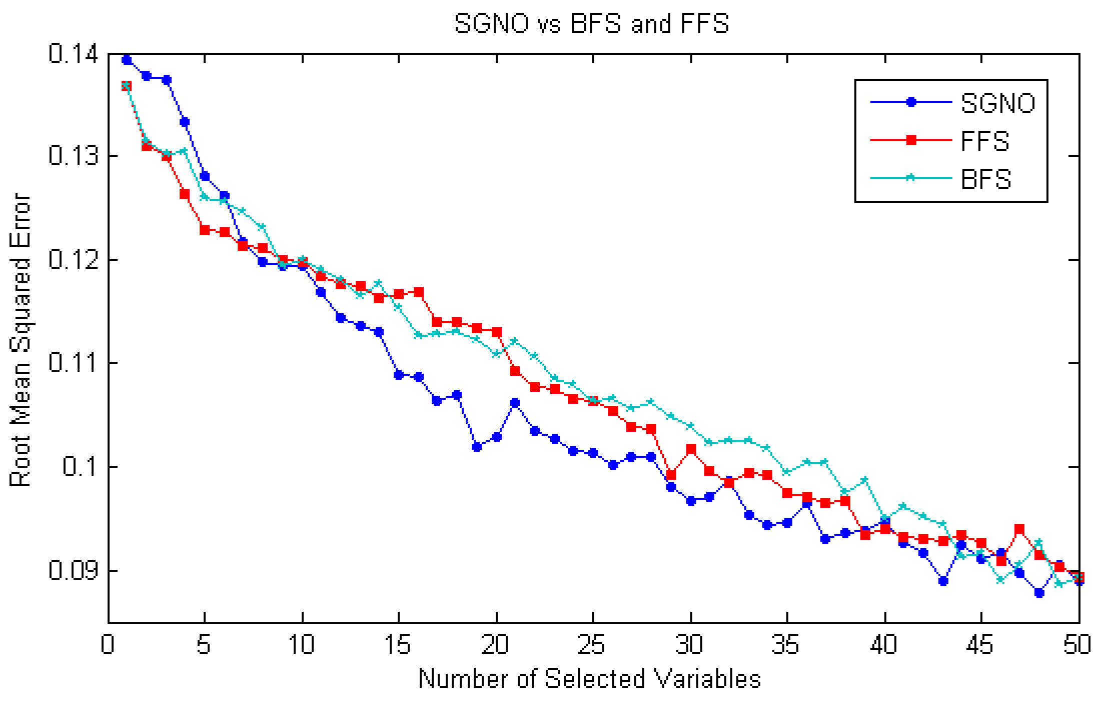

Figure 4.16: Performance of NN models using BFS and FFS against SGNO

As shown in Figure 4.16, all these plots show gradually decreasing patterns. The plots for BFS and FFS are of similar patterns and when the number of selected variables is greater than 26, the performance of FFS selected variables is slightly better, i.e. lower RMSEs. SGNO performs the worst when the number of variables is less than 6. However, as the number of variables increases, SGNO gradually gains its advantage and generates the lowest RMSEs when the number of variables is between 10 and 40. When the number of selected variables exceeds 40, these performance plots start to overlap.

4.8.3 Genetic Neural Mathematical Method approach

The GNMM is a hybrid data mining technique implemented by Yang (2007, 2010) using GA and NNs. The selection of variables, i.e. the construction of the importance rank table, is performed using the appearance percentage of each individual variable during the GA process. The detailed procedures of GNMM are discussed in section 3.3.3. Figure 4.17 illustrates the appearance percentage for the variables in this tomato prediction application.

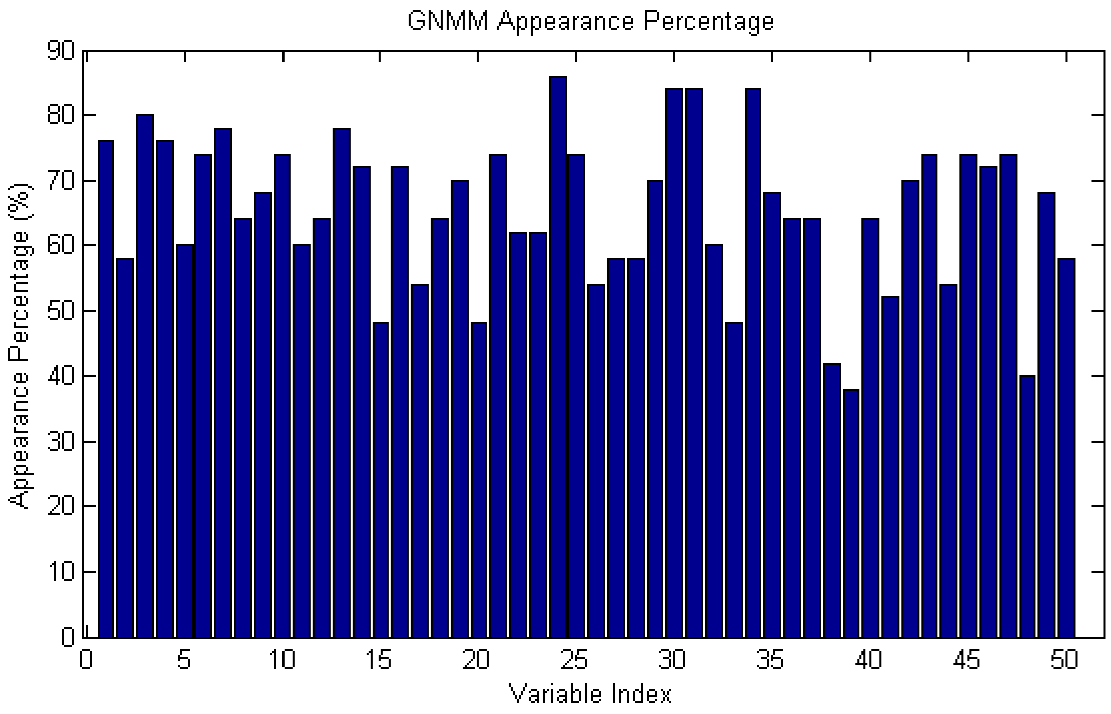

Figure 4.17: GNMM Appearance Percentage

Rearranging the variables in descending order based on their appearance percentages as derived by GNMM, the following rank table was constructed. This rank table is distinct from the one produced by SGNO.

[24, 30, 31, 34, 3, 7, 13, 1, 4, 6, 10, 21, 25, 43, 45, 47, 14, 16, 46, 19, 29, 42, 9, 35, 49, 8, 12, 18, 36, 37, 40, 22, 23, 5, 11, 32, 2, 27, 28, 50, 17, 26, 44, 41, 15, 20, 33, 38, 48, 39]

Figure 4.18 illustrates the appearance frequencies of each variable group (CO_2, radiation, temperature, etc.) in the rank table. This figure clearly shows that the CO_2 group is the most important variable group in general.

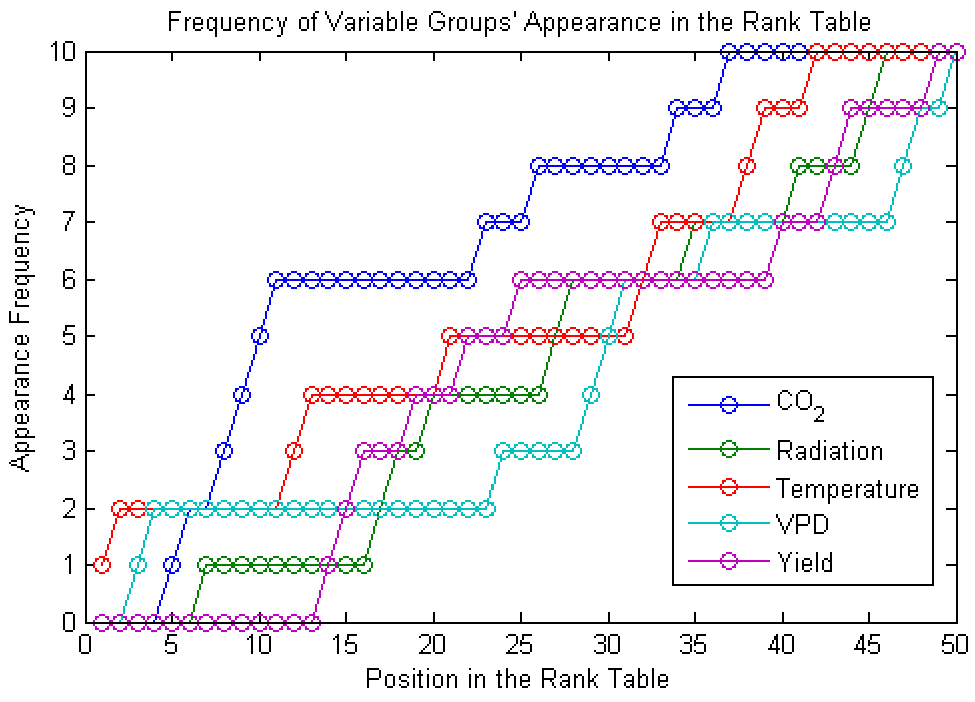

Figure 4.18: Appearance Frequencies of Variable Groups

Table 4.9 lists the performance of GNMM by evaluating various number of variables selected the rank table above using NNs.

Table 4.9: GNMM evaluation results

Size	1	2	3	4	5	6	7	8	9
RMSE	0.1747	0.1628	0.1573	0.1577	0.1468	0.1446	0.133	0.1295	0.1286

Size	10	11	12	13	14	15	16	17	18
RMSE	0.1317	0.1295	0.1238	0.125	0.1225	0.1226	0.1205	0.1189	0.1158

Size	19	20	21	22	23	24	25	26	27
RMSE	0.1185	0.1169	0.1134	0.1157	0.1133	0.1139	0.1125	0.1106	0.1089

Size	28	29	30	31	32	33	34	35	36
RMSE	0.1112	0.1058	0.1066	0.1094	0.1059	0.1032	0.1033	0.1015	0.1006

Size	37	38	39	40	41	42	43	44	45
RMSE	0.0963	0.101	0.0947	0.0993	0.0946	0.0982	0.0978	0.0945	0.0925

Size	46	47	48	49	50
RMSE	0.0904	0.0899	0.0931	0.0892	0.0903

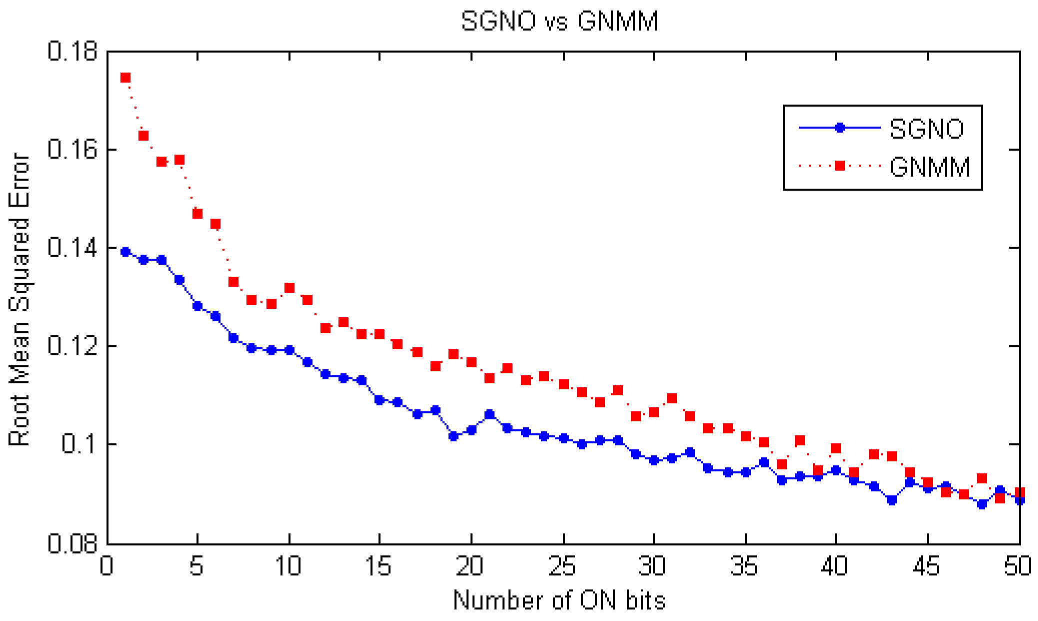

Figure 4.19: Performance of GNMM vs. SGNO

Figure 4.19 illustrates the average performance of GNMM as compared to the variables selected by SGNO under the same conditions (the same of variables and the same NN architectures). This figure indicates that the rank table produced by SGNO performs better than the rank table generated by GNMM. The advantage of SGNO is significant when the number of input variables is less than 7 and the largest difference is up to the value of 0.0355 (1 input variable). As the number of input variables increases, the performance difference between SGNO and GNMM gradually reduces and the performance plots start to overlap when the number of variables exceeds 43.

4.8.4 Benchmarking summary

Figure 4.20 summarises the performance of all the benchmarking techniques against SGNO by plotting the RMSEs produced using the preference lists. As shown in the figure, all plots show gradually decreasing patterns as the number of selected variables increases. When the number of variables is small, less than 10, PCA and GNMM produce the highest RMSEs among these plots, while BFS, FFS and SGNO generate significantly lower RMSEs. As the number of variables increases, all plots are getting closer indicating that BFS, FFS and SGNO start losing their advantages and all techniques managed to include favourable variables in the evaluated variable lists. Among all these techniques, BFS, FFS and SGNO are superior to PCA and GNMM. Furthermore, SGNO is of similar performance against BFS and FFS when the number of variables is less than 10, and becomes superior to BFS and FFS thereafter.

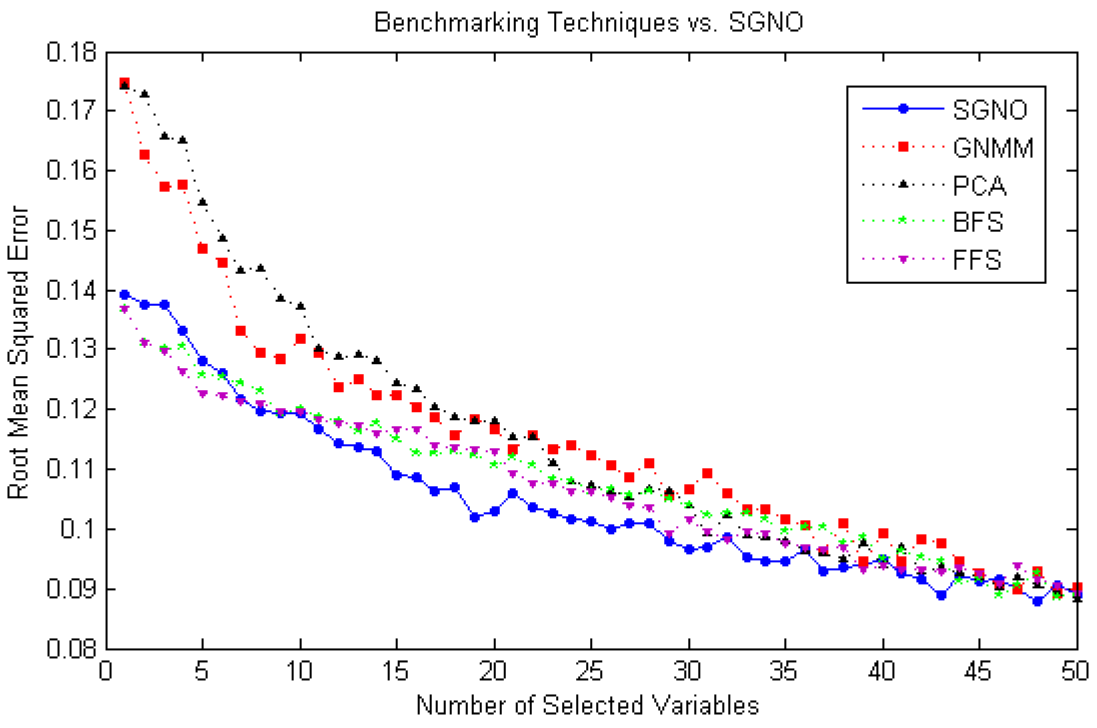

Figure 4.20: Performance of all benchmarking techniques against SGNO

4.9 Conclusion

In this chapter, SGNO is applied to a dataset consisting of environmental conditions and tomato weekly yields collected between 1999 and 2007 in a greenhouse. The environmental conditions include CO_2 density, solar radiation level, temperature and VPD inside the greenhouse. The purpose of SGNO in this application is to establish NN models to estimate the tomato weekly yields based on the environmental conditions and weekly yields data collected previously, and identifies the influences of those data in the established model. Apart from producing the estimation model, the symbolic relationships between the model inputs and output are discovered using a MBE-GP.

Analysis of the SGNO's results shows that almost all the input variables are contributing to the output as increasing the number of input variables would generally increase the accuracy of estimation. Among the five variable groups, radiation is the most important as it always has higher appearance frequencies than other variable groups (see Figure 4.8). The models established are capable of producing average errors within the range (0.09, 0.14) depending on the number of input variables and the conditions (network architecture and initial neural weights) of the NNs. In addition, the discoveries of several 'optimal' NNs with various numbers of input variables show that the prediction errors can be improved significantly (see Figure 4.12).

After identifying the input variables that have the most influence, the MBE-GP successfully constructed the concise relationships (symbolic expressions) between the input variables and the output. The symbolic expressions discovered can predict the tomato weekly yield with accuracies competitive to the NN models with estimated architectures but not as good as the NN models with the 'optimal' architectures.

Comparing with four benchmarking FS techniques (PCA, BFS, FFS and GNMM), SGNO performs better than PCA and GNMM as the prediction errors produced by SGNO are always lower than the errors generated by PCA and GNMM, and the largest difference is up to the value of 0.035. BFS and FFS produce performance patterns similar to SGNO. When the number of input variables is less than 6, SGNO performs the worst comparing to BFS and FFS. However, as the number of variables increases, SGNO gradually gains its advantage and generates the lowest prediction errors (the largest difference is about 0.01 when the number of variables is around 20). When the number of variables becomes large (over 40), all these 5 techniques produce similar level of performance (see Figure 4.20).

References

Adams S.R. (2002) Predicting the Weekly Fluctuations in Glasshouse Tomato Yields. Acta Horticulture 593:19-24.

Adams S.R., Valdes V.M. (2002) The effect of periods of high temperature and manipulating fruit load on the pattern of tomato yields. Journal of Horticultural Science & Biotechnology 77:461-466.

Adams S.R., Cockshull K.E., Cave C.R.J. (2001a) Effect of Temperature on Growth and Development of Tomato Fruits. Annals of Botany 88:869-877.

Adams S.R., Valdes V.M., Cave C.R.J., Fenlon J.S. (2001b) The impact of changing light levels and fruit load on the pattern of tomato yields. Journal of Horticultural Science & Biotechnology 76:368-373.

Berry M.J.A., Linoff G. (2004) Data Mining Techniques: for marketing, sales, and customer relationship management Wiley Publication.

Cox E. (2005) Fuzzy Modeling and Genetic Algorithms for Data Mining and Exporation Morgan Kaufmann, San Francisco.

Csaji B.C. (2001) Approximation with Artificial Neural Networks, Faculty of Science, Eotvos Lorand University. pp. 45.

de Koning A.N.M. (1988) The effect of different day/night temperature regimes on growth, development and yield of glasshouse tomatoes. Journal of Horticultural Science 63:465-471.

de Koning A.N.M. (1990) Long-term temperature integration of tomato growth and development under alternating temperature regimes. Scientia Horticulturae 45:117-127.

Dunteman G.H. (1989) Principal Components Analysis SAGE Publications.

Fitz-Rodriguez E., Giacomelli G.A. (2009) Yield Prediction and Growth Mode Characterization of Greenhouse Tomatoes with Neural Networks and Fuzzy Logic. Transactions of the ASABE 52:2115-2128.

Heaton J.T. (2005) Introduction to Neural Networks with Java Heaton Research.

Hornik K., Stinchcombe M., White H. (1989) Multilayer feedforward networks are universal approximators. Neural Networks 2:359-366.

Hoshi T., Sasaki T., Tsutsui H., Watanabe T., Tagawa F. (2000) A daily harvest prediction model of cherry tomatoes by mining from past averaging data using topological case-based modeling. Computers and electronics in agriculture 29:149-160.

Kano A., van Bavel C.H.M. (1988) Design and Test of a Simulation Model of Tomato Growth and Yield in a Greenhouse. Journal of the Japanese Society for Horticultural Science 56:408-416.

Kaul M., Hill R.L., Walthall C. (2005) Artificial neural networks for corn and soybean yield prediction. AGricultural Systems 85:1-18.

Kehagias A., Panagiotou H., Maslaris N., Petridis V., Petrou L., Spais V. (1998) Predictive modular neural network methods for prediction of sugar beet crop yield, IFAC Conference of Control Applications and Ergonomics in Agriculture, Athens, Greece.

Leonardi C., Guichard S., Bertin N. (2000) High vapour pressure deficit influences growth, transpiration and quality of tomato fruits. Scientia Horticulturae 84:285-296.

Li J., Zhou J.M., Duan Z.Q., Du C.W., Wang H.Y. (2007) Effect of CO_2 Enrichment on the Growth and Nutrient Uptake of Tomato Seedlings. Pedosphere 17:343-351.

McAvoy R.J. (1989) The Effect of Total Available Photosynthetic Photon Flux on Single Truss Tomato Growth and Prediction. Journal of Horticultural Science 64:331-338.

Mulholland B.J., Edmondson R.N., Fussell M., Basham J., Ho L.C. (2003) Effects of high temperature on tomato summer fruit quality. Journal of Horticultural Science & Biotechnology 78:365-374.

Nilsen S., Hovland K., Dons C., Sletten S.P. (1983) Effect of CO_2 enrichment on photosynthesis, growth and yield of tomato. Scientia Horticulturae 20.

O'Kane P.C. (1973) Forecasting Plant Development Time in the Tomato Production Industry: A Sugested Simulation Model. Irish Journal of Agricultural Economics and Rural Sociology 4:77-87.

Pandey A., Srivastava J.K., Rajput N.S., Prasad R. (2010) Crop parameter estimation of Lady finger by using different neural network training algorithms. Russian Agricultural Sciences 36:71-77.

Prasad A.K., Chai L., Singh R.P., Kafatos M. (2006) Crop yield estimation model for Iowa using remote sensing and surafce parameters. International Journal of Applied Earch Observation and Geoinformation 8:26-33.

Simpson G. (1994) Crop yield prediction using a CMAC neural network, in: J. Desachy (Ed.), Image and Signal Processing for Remote Sensing, SPIE. pp. 160-171.

Stoikos G. (1995) Sugar beet crop yield prediction using artificial neural networks (in Greek), Modern Technologies Conference in Automatic Control, Athens, Greece. pp. 120-122.

Tang W.C. (1995) The Prediction and Validation of Greenhouse Tomato Yield using Mathematical Models and Expert Systems, Bio-Resource Engineering, The University of British Columbia. pp. 148.

Willits D.H., Peet M.M. (1998) The effect of night temperature on greenhouse grown tomato yields in warm climates. Agricultural and Forest Meteorology 92:191-202.

Zhang F., Iliescu D.D., Hines E.L., Leeson M.S., Adams S.R. (2010) Decision Support System for Greenhouse Tomato Yield Prediction using Artificial Intelligence Techniques, in: B. Manos, et al. (Eds.), Decision Support Systems in Agriculture, Food and the Environment: Trends, Applications and Advances, IGI Global.

Zhang Y.X., Li H., Hou A.X., Havel J. (2006) Artificial neural networks based on principal component analysis input selection for quantification in overlapped capillary electrophoresis peaks. Chemometrics and intelligent laboratory systems 82:165-175.

CHAPTER 5

Prediction of Longitudinal Dispersion Coefficient in Natural Streams

5.1 Overview

In the previous chapter, SGNO was used to select input variables for the NN models that predict the weekly yields of tomatoes in greenhouses. This chapter demonstrates the application of SGNO to optimise NN models for predicting the longitudinal dispersion coefficient in natural rivers.

River contamination/pollution has received great attention in recent decades. The contaminant being spilled accidentally or intentionally into the water supply is a constant concern for those using it. Travel time and mixing of water are the basic stream flow characteristics that should be understood in order to study the rate of stream movement and pollution dilution in case it is introduced into streams (Jobson, 1996). Dispersion, in hydrodynamic terms, is the spreading of solutes from highly concentrated areas to areas of less concentration in flowing fluid. Dispersion generally takes place along all three dimensions of the stream and these dispersions are known as longitudinal dispersion, which takes place along the stream flow, transversal dispersion, which refers to the mixing of solutes towards banks, and vertical dispersion, which takes place towards the stream bed and surface (if applicable) (Jobson, 1996; Veliskova et al., 2009). When soluble substances are discharged into a river, the current of the stream transports the substance downstream while the substance mixes with the stream. In the early stages, advection is the most important hydrodynamic behaviour in the transportation of the solute. Vertical dispersion generally completes rapidly within the distance of a few river-depths and transversal dispersion is much slower but usually completes within a few kilometres (Ahsan, 2008). In the later stages, longitudinal dispersion, which continues indefinitely without boundaries, becomes the important mixing process when the cross-sectional (vertical and transversal) dispersions completes (Jobson, 1996). The longitudinal dispersion coefficient, a measure of the rate of mixing of contaminants in rivers, is an important fundamental parameter in hydrodynamic modelling. Knowledge of the accurate value of longitudinal dispersion coefficient is essential in solving problems of pollution transport and modelling water quality (Sahay, 2010; Veliskova et al., 2009).

5.2 Background

It has been found that the dispersion process of pollutants in natural rivers and streams is complicated. Many hydrodynamic and geometrical parameters affect the longitudinal dispersion coefficient, especially in natural rivers, including the stream velocity, bed configuration, secondary flow, and many other factors. Hence the longitudinal dispersion coefficients will vary under different stream characteristics especially in different rivers.

In general, direct measurement of longitudinal dispersion coefficients by measuring the concentrations of contaminant samples at various locations of a river is time consuming. For those cases when the dispersion data is not available, alternative methods, including theoretical and empirical regression models, are developed to estimate the longitudinal dispersion coefficients (Deng et al., 2002; Kashefipur and Falconer, 2002; Sahay, 2010).

Back in 1954, Taylor (1954) introduced the term 'longitudinal dispersion coefficient' and proposed his study of the dispersion of soluble matters in laboratory-based pipes. Elder (1959) expanded Taylor's studies and derived the well-known equation

$$\boldsymbol{D_l = \left(\frac{0.4041}{k^3} + \frac{k}{6}\right) HU_* \;\; or \;\; D_l = 5.93HU_*} \tag{5.1}$$

where D_l represents the longitudinal dispersion coefficient, k is the von Karman constant (approximately equal to 0.41), H is the depth of flow and U_* is the shear-stress velocity.

Elder's equation has been widely used due to its simplicity. However, Elder's equation does not accurately describe the longitudinal dispersion in natural rivers and significantly underestimates the longitudinal dispersion in real streams as the equation is derived based on laboratory measurements (Fischer et al., 1979; Kashefipur and Falconer, 2002; Seo and Baek, 2004). Using the lateral velocity profile, Fischer (1967) presented a new integral equation for D_l

$$\boldsymbol{D_l = -\frac{1}{A}\int_0^W hu' \int_0^y \frac{1}{\varepsilon_t h} \int_0^y hu' \, dydydy} \tag{5.2}$$

where D_l is the longitudinal dispersion coefficient, A is the cross-sectional area of the stream, W is the channel width, h is the local depth of flow, u' is the deviation of the velocity from the cross-sectional mean velocity, y is the Cartesian coordinate measured from the left bank and ε_t is the transverse mixing coefficient (Fischer, 1967; Seo and Baek, 2004).

Equation 5.2 is rather difficult to use as elaborate transverse velocity profiles and cross-sectional geometry are not readily available to the field engineers. Thus, Fischer (1979) developed a simpler equation by replacing the triple integration, velocity deviation and transverse mixing coefficient with reasonable approximations

$$\boldsymbol{D_l = \frac{0.011U^2W^2}{HU_*}} \tag{5.3}$$

where D_l is the longitudinal dispersion coefficient, U is the cross-sectional mean velocity, W is the channel width, H is the cross-sectional mean depth of flow and U_* is the shear-stress velocity (Seo and Baek, 2004).

Since 1976, several researchers have presented empirical or experimental equations to express longitudinal dispersion coefficient as a function of hydraulic and geometric parameters. The general form of the equations can be expressed as

$$\boldsymbol{D_l = a(HU_*)\left(\frac{U}{U_*}\right)^b\left(\frac{W}{H}\right)^c} \tag{5.4}$$

where a, b and c are constants proposed by the researchers (Seo and Baek, 2004).

Table 5.1 lists some of the constants proposed by some researchers

Table 5.1: List of constants in the proposed equations

Proposed Equation	a	b	c
Fischer (1979)	0.011	2.0	2.0
Seo and Cheong (1998)	5.92	1.43	0.62

Koussis and Rodriguez-Mirasol (1998)	0.6	0.0	2.0

In recent years, researchers have successfully applied artificial intelligence techniques to a number of multivariate forecasting problems in hydrology. Rowinski, Piotrowski and Napiorkowski (2005) employed NNs to estimate longitudinal dispersion coefficient using channel depth and width, cross-sectional mean flow velocity, shear velocity and sinuosity index. Their work demonstrated that conventional feed-forward back-propagation (FFBP) NNs are promising. Tayfur and Singh (2005) developed five NN models taking various geometric and hydraulic parameters as model inputs. The best performing model in their work achieved up to 90% accuracy and used mean channel depth, channel width and mean flow velocity as input parameters. Yang et al. (2007) proposed a novel hybrid method, GNMM, utilising GA and NNs. They determined that the daily mean flow rate and instant flow rate at the gauging stations are the most important factors and their regression model can predict the longitudinal dispersion coefficient with a determination coefficient (R^2) up to the value of 0.72. Toprak and Cigizoglu (2008) employed three different types of NNs and showed that FFBP NN generates the best performance. Sahay (2010) studied the performance of NNs with various numbers of input parameters and showed that the combination of channel width, channel depth, mean cross-sectional flow velocity, shear velocity and sinuosity gives the best performance with up to 65% accuracy.

Apart from the use of NNs, Toprak, Savcl and Avcl (2004) demonstrated that the application of fuzzy logic has significant advantage over the empirical models. In recent works by Azamathulla and Wu (2011) and Azamathulla and Ghani (2010) applied Support Vector Machines (SVMs) and Genetic Programming (GP) respectively to estimate longitudinal dispersion coefficient using channel width, channel depth, mean flow velocity and shear velocity. The results show that SVM can produce RMSEs as low as 0.0078 with a corresponding correlation coefficient between the predictions and the actual values of up to 0.95; and GP is capable of producing excellent predictions with RMSE as low as 0.085 and correlation coefficient up to 0.99.

5.3 Dataset

In the last decade, the UK Environment Agency (EA) has completed a great number of dye tracing experiments to study travel time-flow relationships and the dispersive characteristics. Dye tracing studies were carried out at 27 different rivers in the EA regions and a total of 196 data samples were collected at various gauging stations. Each data sample contains geographical, geometrical and hydraulic parameters including the river's location, catchment area, reach length, discharge rate, etc. The data samples are stored in a database developed by Guymer (1999) in a standardised data storage format. Yang (2010) demonstrated his novel processing algorithm using this database and discovered that the daily mean flow rate and instant flow rate measured at the gauging stations are the most valuable parameters to predict the longitudinal dispersion coefficient. In addition, his algorithm achieved a good correlation coefficient of up to 0.85

between the estimated longitudinal dispersion coefficient and the observed values.

The dataset contains 196 samples and each sample contains a total of 71 parameters (text and numeric). Among the 196 samples, a certain number of samples are considered invalid as some of the parameters are missing and replaced by zeros. These invalid samples are omitted in the processing and hence the validated dataset contains only 127 samples. For these 127 valid samples, each sample consists of 49 numeric parameters and a number of text parameters, only the numeric parameters are of interest as the SGNO can only handle numeric variables. Among these 49 numeric parameters, 21 parameters including the directly measured parameters and those, which have been proved to be effective parameters for the estimation of longitudinal dispersion coefficients, are selected. Table 5.2 lists these 21 parameters to be studied.

Table 5.2: Selected parameters to be studied

Variable Group	Variable Name	Symbol	Unit
Injection Point	Theoretical mean flow	M_i	m^3/s
	Theoretical Q95 flow	Q_i	m^3/s
Start Location	Catchment area	C_s	m^2
	Distance from injection point	D_s	m
	Theoretical mean flow	M_s	m^3/s
	Theoretical Q95 flow	Q_s	m^3/s
End Location	Catchment area	C_e	m^2
	Distance from injection point	D_e	m
	Theoretical mean flow	M_e	m^3/s
	Theoretical Q95 flow	Q_e	m^3/s
Reach	Reach length	L	m

	Straight distance	D_r	m
	Slope	S_1	–
	Sinuosity	S_2	–
Gauging Station	Catchment area	C_g	m^2
	Theoretical mean flow	M_g	m^3/s
	Theoretical Q95 flow	Q_g	m^3/s
	Instantaneous flow	F_g	m^3/s
	Tracer mean travel time	T	s
	Tracer mean velocity	V	m/s
	Longitudinal dispersion coefficient	K	m^2/s

Table 5.3 summarises the statistics of these variables. Of these 21 selected parameters, the longitudinal dispersion coefficient, *K*, is the output of the NN models used in SGNO, while the other parameters are considered as the possible input variables for the NN models, based on which the output variable, *K*, is derived.

Table 5.3: Statistics of the selected parameters

Index	**Symbol**	**Mean**	**Minimum**	**Maximum**	**Standard Deviation**
1	M_i	7.96	0	40.61	8.89
2	Q_i	1.34	0	8.15	1.72
3	C_s	568.23×10^6	0	3314.75×10^6	677.59×10^6
4	D_s	9824.02	1000	41500	7141.03
5	M_s	9.86	0	49.55	11.48
6	Q_s	1.59	0	9.47	1.97
7	C_e	858.97×10^6	9.25×10^6	3315.25×10^6	738.83×10^6
8	D_e	16426.61	3400	46500	8726.89
9	M_e	11.33	0	49.55	12.70
10	Q_e	1.83	0	9.47	2.25

11	L	6037.06	1058	14697	2923.17
12	D_r	4342.88	915.57	12133.5	1970.32
13	S_1	2.33×10^{-3}	0.01×10^{-3}	24.37×10^{-3}	3.12×10^{-3}
14	S_2	1.40	1.02	2.91	0.34
15	C_g	792.39×10^6	20×10^6	3314.8×10^6	681.34×10^6
16	M_g	12.38	0.444	47.136	10.54
17	Q_g	1.93	0.06	6.6	1.51
18	F_g	10.20	0.5	75	12.97
19	T	37.04×10^3	3.33×10^3	317.78×10^3	43.58×10^3
20	V	0.26	0.03	1.04	0.18
21	K	10.80	0.04	72.46	11.94

5.4 Data Pre-processing

In this application, the parameters are rescaled using the mean-standard deviation approach due to the relatively small size of the dataset and the actual scale (possible range) of each variable is unknown. Hence, min-max normalisation is impractical for this dataset. The 127 samples are divided randomly into five groups to apply the five-fold cross-validation (see section 3.2.2.3.2). Table 5.4 shows the distributions of these five groups.

Table 5.4: Sample distributions in five-fold cross-validation

	Set A	**Set B**	**Set C**	**Set D**	**Set E**	**Total**
Count	25	26	25	26	25	127

5.5 Variable Selection using Sensitivity Genetic Neural Optimisation (SGNO)

In this application, SGNO is applied to the selected numeric dataset to determine what, if any, influence the 20 input parameters have on the estimation of the output variable, the longitudinal dispersion coefficient. The following sections describe the SGNO process as applied to this dataset.

5.5.1 Neural Network module

The NN module in SGNO is designed to evaluate the performance of the variables/parameters selected by the chromosome in the GA module as the inputs of the NN module. In SGNO, the NN module employs five-fold cross-validation with early stopping; the number of hidden layers in the NN module is fixed to 1; the number of hidden neurons is estimated by halving the sum of input and output variables (Equation 3.5), and the initial weights of the neurons are random numbers. In addition, the tangent-sigmoid function (Equation 2.5) is used as the transfer functions in the hidden neurons and the pure linear function (Equation 2.2) is used in the neuron in the output layer. The application of the pure linear transfer function in the output layer is due to the fact that the target output of the NN module, the rescaled longitudinal dispersion coefficient, lies inside the range [-0.90, 5.17] and thus a transfer function with output range covering this range is required.

5.5.2 Genetic Algorithm module

In this application, the size of each binary chromosome in the GA module is 20 as there are 20 parameters to be analysed. The size of the GA population is estimated at 100 using the rule of thumb discussed previously in section 3.2.2.2 and equation 3.3. The size of the elite group containing chromosomes passed to the new generation without modification is 5, which is 5% of the total population. Furthermore, the GA module is coded to stop after evaluating 50 generations.

Figure 5.1 illustrates the performance of the chromosomes in each generation. It is clear in the figure that the mean RMSE reduces rapidly in the first few generations and shows a slowly decreasing pattern in the following generations. The mean RMSE also shows a trend of convergence since approximately the 15th generation. In addition, the range of estimated errors in each generation shrinks as the GA module evolves.

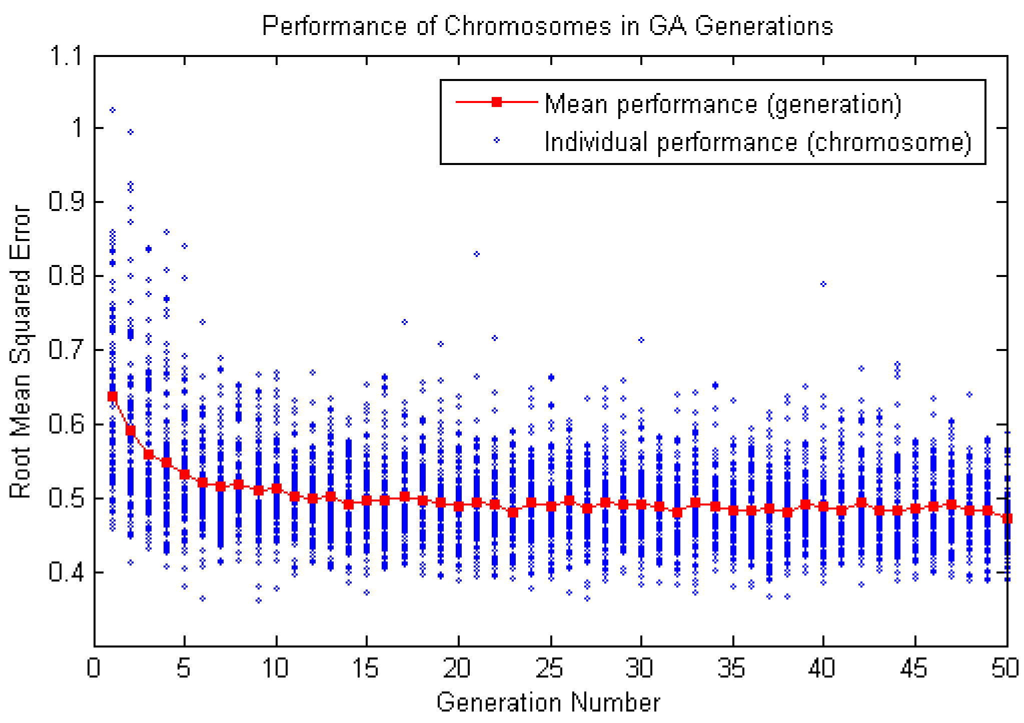

Figure 5.1: Chromosomes performance in GA generations

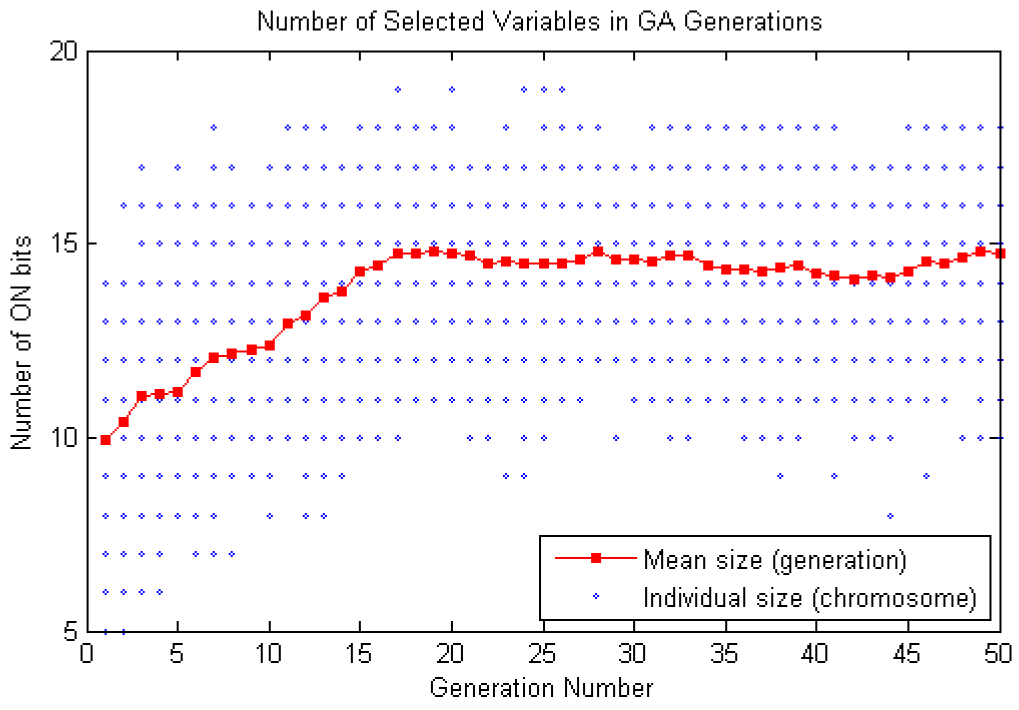

Figure 5.2: Number of selected variables (ON bits) in each generation

Figure 5.2 illustrates the size of chromosomes (the number of ON bits) evaluated in the GA generations. The mean size of the chromosomes in each generation shows a clear increasing trend in the first 17 generations followed by a roughly converging pattern between 14 and 15 bits.

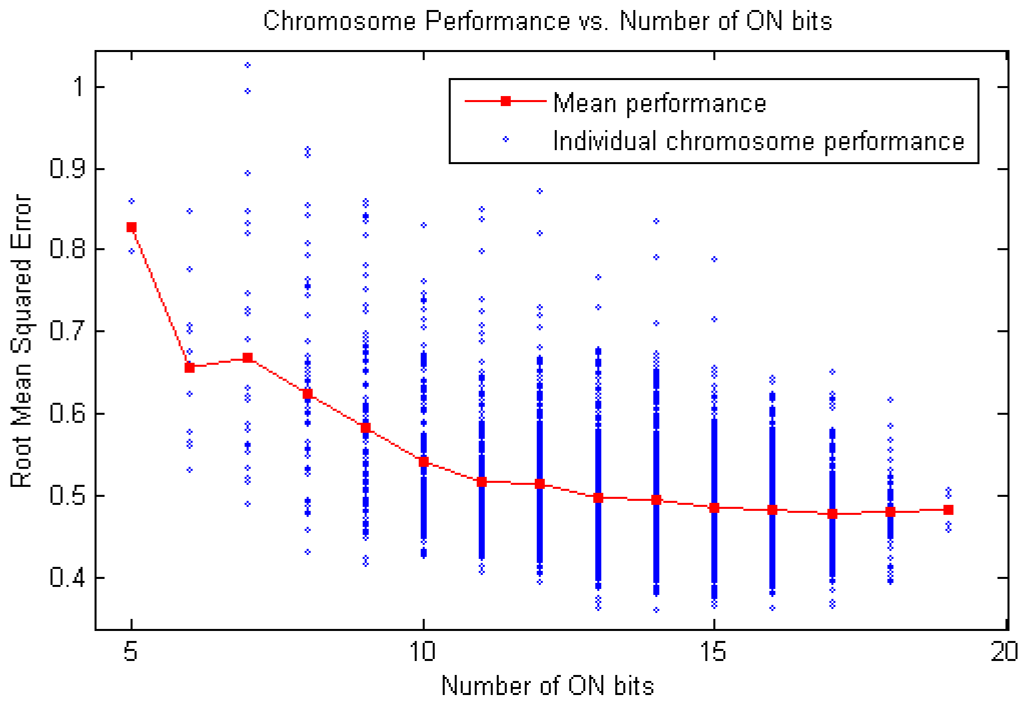

Figure 5.3: Chromosome Performance vs. Number of Selected Variables

Figure 5.3 illustrates the sizes of all chromosomes evaluated during the GA process and their corresponding performance represented by RMSE. The figure clearly shows that the mean RMSE decreases as the sizes of the chromosomes increase and the mean RMSE starts to converge when the size of the chromosome exceeds 13.

5.5.3 Sensitivity Analysis module

The SA module is designed to analyse the global importance of the parameters appearing in the group of 'well-performing' chromosomes. The group is formed by selecting a quarter of the chromosomes, which have the highest fitness (the lowest MSE), from each generation in the GA process. Each chromosome in the selected group, together with its associated NNs, is considered to be an effective model representing the relationship between the input parameters and the output parameter. The input parameters in a model are of different levels of importance and SA is capable of quantifying the importance. For each selected chromosome, the sensitivity scores of the variables (represented by the ON bits) in that chromosome are calculated. The global sensitivity score of a variable, i.e. the importance/influence of that variable, is derived by taking the mean value of its sensitivity scores in all selected chromosomes.

In this application, the GA module evaluated a population of 100 chromosomes for 50 generations and thus the total amount of chromosomes evaluated is 5000. A quarter of all chromosomes are selected to carry out the SA analysis. Figure 5.4 illustrates the size (number of ON bits) of the selected chromosomes. The figure clearly shows that most of the selected chromosomes are of large sizes (the number of ON bits is greater than or equal to 14 out of 20). The existence of chromosomes which have a small number of ON bits (less than 10) indicates that the models using a small number of input parameters can produce good performance, which is competitive to the best-performing models.

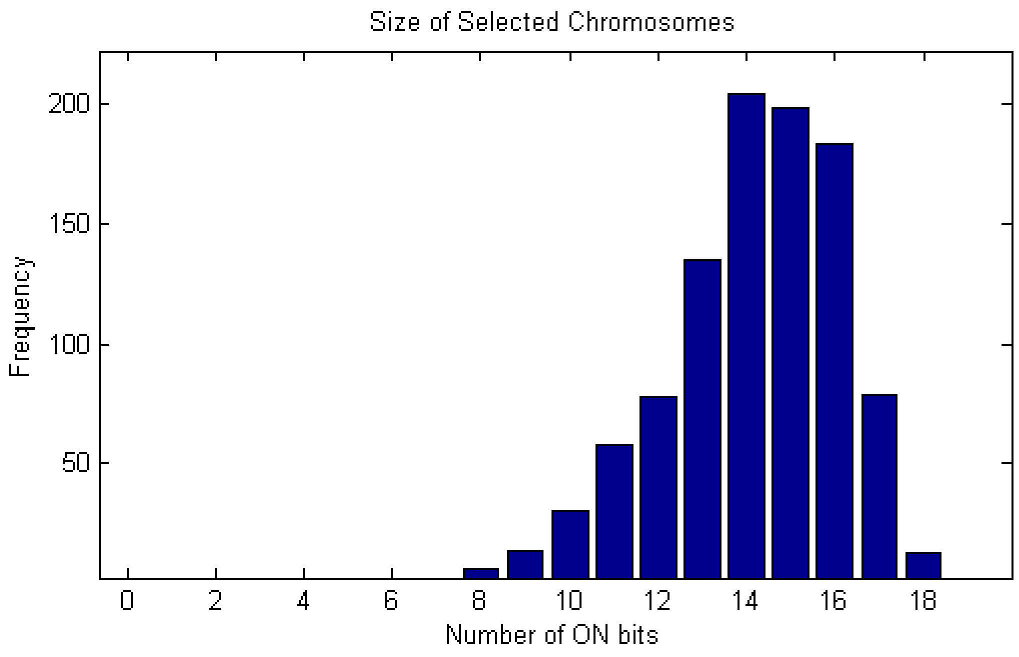

Figure 5.4: Size of the chromosomes selected for SA analysis

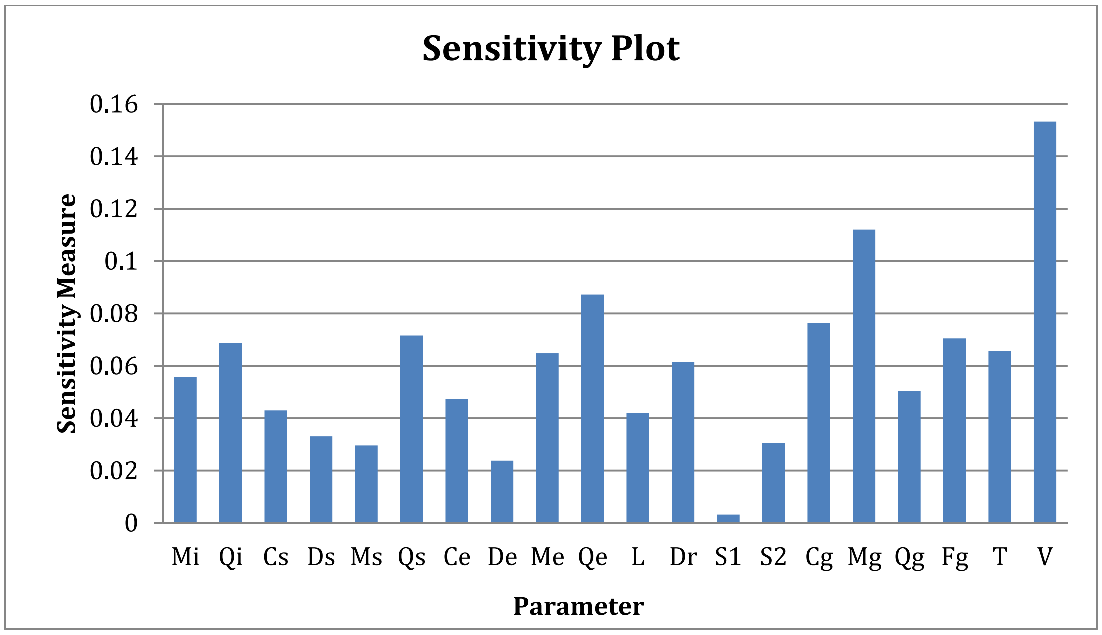

Figure 5.5: Global sensitivity scores of all parameters

Figure 5.5 illustrates the global sensitivity scores for all the input parameters. In the figure, the mean velocity of the tracer (V) and the mean flow rate at the gauging station (M_g) stand out, while the reach slope has the lowest sensitivity score. This indicates that the mean velocity of the tracer (*V*) is the most influential parameter in the model and the reach slope (S_1) is the least influential parameter. By rearranging the parameters according to their global sensitivity score, the parameter importance rank table can be constructed. The following list is the rank table containing the parameters in descending order, in which the former elements have higher sensitivity scores than the latter elements, and hence the former elements are more important.

[V, M_g, Q_e, C_g, Q_s, F_g, Q_i, T, M_e, D_r, M_i, Q_g, C_e, C_s, L, D_s, S_2, M_s, D_e, S_1]

In terms of the indices of the chromosomes in the GA module, the rank table can be rewritten as follows:

[20, 16, 10, 15, 6, 18, 2, 19, 9, 12, 1, 17, 7, 3, 11, 4, 14, 5, 8, 13]

5.6 System Remodelling

As discussed in 3.2.3, NNs are not only used to evaluate the chromosomes in the GA module, but are also used to re-evaluate the performance of parameters in the sensitivity rank table. Table 5.5 lists the performance of NN models using various numbers of input parameters selected from the SGNO's importance rank table; the selection of parameters always start with the first element in the rank table, e.g. [1], [1, 2], [1, 2, 3], [1, 2, 3, 4], and so on. The 'Size' rows indicate the number of variables selected from the rank table. Figure 5.6 illustrates the performance of NN models taking 3, 6 and 9 input parameters. Figure 5.7 illustrates the performance of these NN models against the mean performance of the chromosomes evaluated in the GA process.

Table 5.5: SGNO re-evaluation results

Size	1	2	3	4	5	6	7	8	9
RMSE	0.6439	0.5918	0.606	0.591	0.5618	0.5559	0.5256	0.5317	0.5221
R^2	0.5821	0.647	0.6298	0.6479	0.6818	0.6885	0.7215	0.715	0.7252

Size	10	11	12	13	14	15	16	17	18
RMSE	0.4835	0.4798	0.4654	0.4744	0.467	0.4764	0.4817	0.4771	0.4782
R^2	0.7644	0.768	0.7817	0.7732	0.7802	0.7713	0.7661	0.7706	0.7695

Size	19	20
RMSE	0.4767	0.4879
R^2	0.771	0.7601

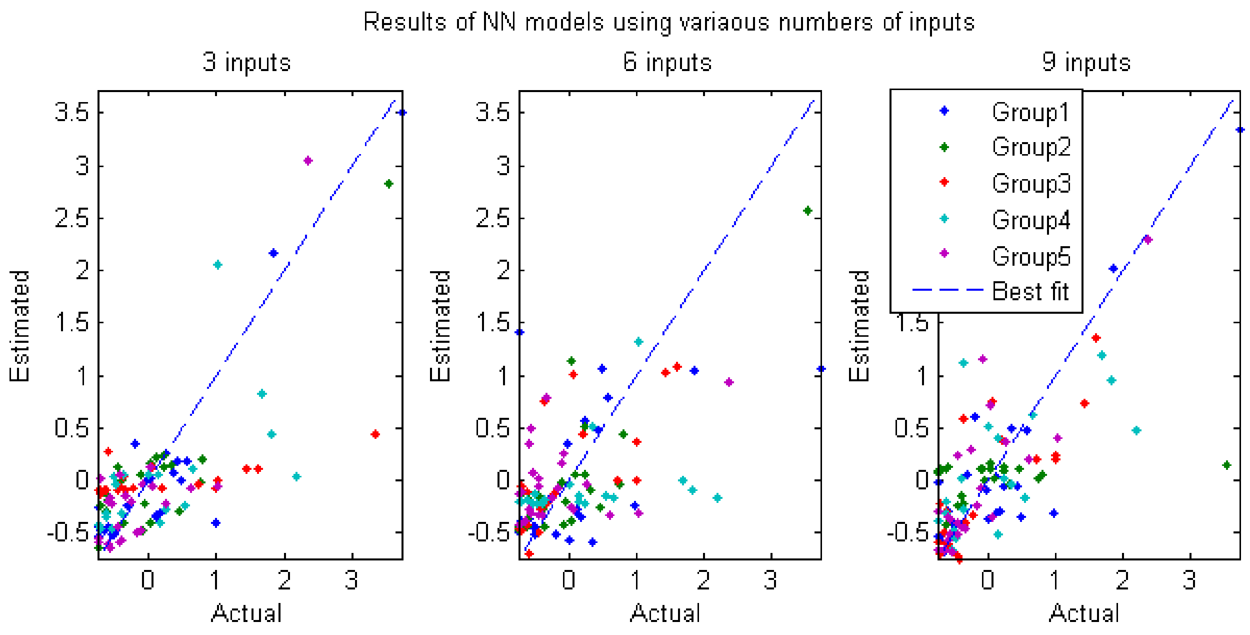

Figure 5.6: Performance of NN models using 3, 6 and 9 inputs

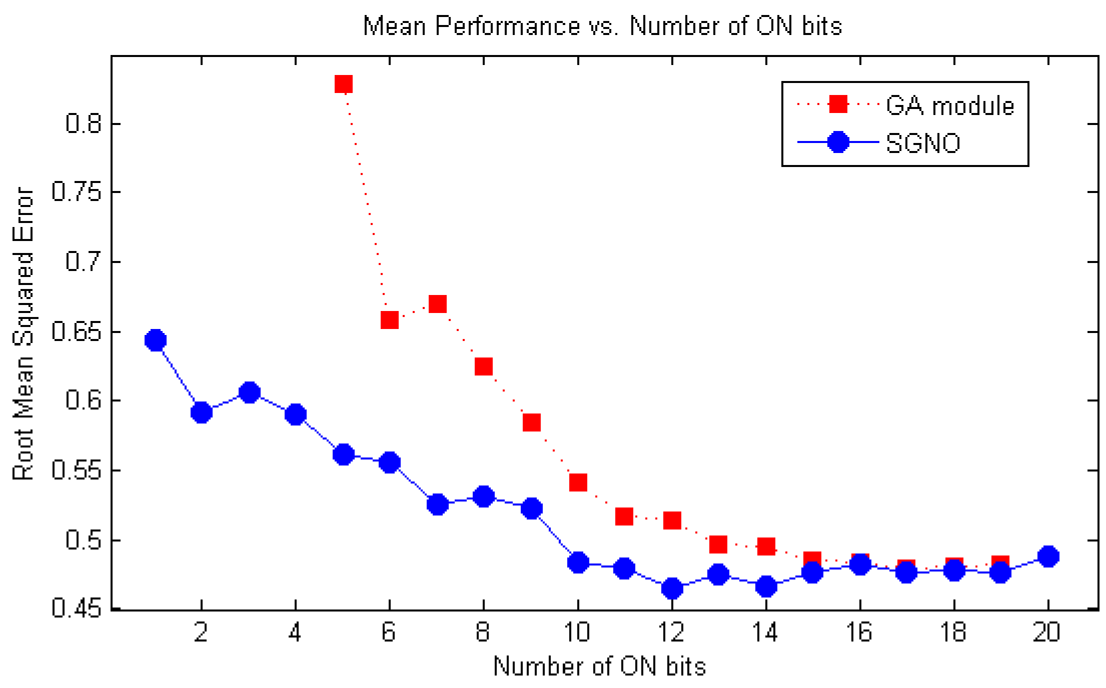

Figure 5.7: Performance of parameters in SGNO rank table against parameters in GA module

In Figure 5.7, both plots clearly show a decreasing pattern as the number of ON bits increase, though there are some points which do not follow this general trend. Significant performance differences between the parameters selected in the SGNO rank table and the parameters evaluated in the GA module are presented clearly, especially when the number of ON bits is small, say less than 10. The potential advantage of SGNO weakens as the number of ON bits increases until it reaches 15, beyond which the performance plots overlap.

The NNs used so far only have one hidden processing layer and the number of hidden neurons is estimated by halving the number of inputs and outputs using Equation 3.5. This architecture estimation may not be optimal, the most popular approach to find the best architecture is by trial and error (Ahmed, 2005). In this application, several NNs of 'optimal' architectures using various numbers of input variables (3 variables, 6 variables, 9 variables, 12 variables and 15 variables) are discovered to find the best performance achievable using NN models. The forward selection procedure (see Section 3.2.3) is implemented to find the best architecture, which gradually increases the model complexity (number of hidden layers and number of hidden neurons) until the following conditions are met (Heaton, 2005):

- The number of hidden neurons should be no more than twice the input layer size.
- There should be no more than two hidden layers.

Table 5.6 lists the 'optimal' architectures discovered for various numbers of parameters selected from the SGNO rank table and the performance of the corresponding models. Figure 5.8 illustrates the performance differences between the 'optimal' architectures and those evaluated in SGNO.

Table 5.6: 'Optimal' architectures and their performance

Number of input variables	**'Optimal' architecture [1st layer, 2nd layer]**	**Performance (RMSE)**
3	[6, 0]	0.5547
6	[11, 0]	0.4646
9	[14, 0]	0.4652
12	[11, 11]	0.4371
15	[20, 8]	0.4388

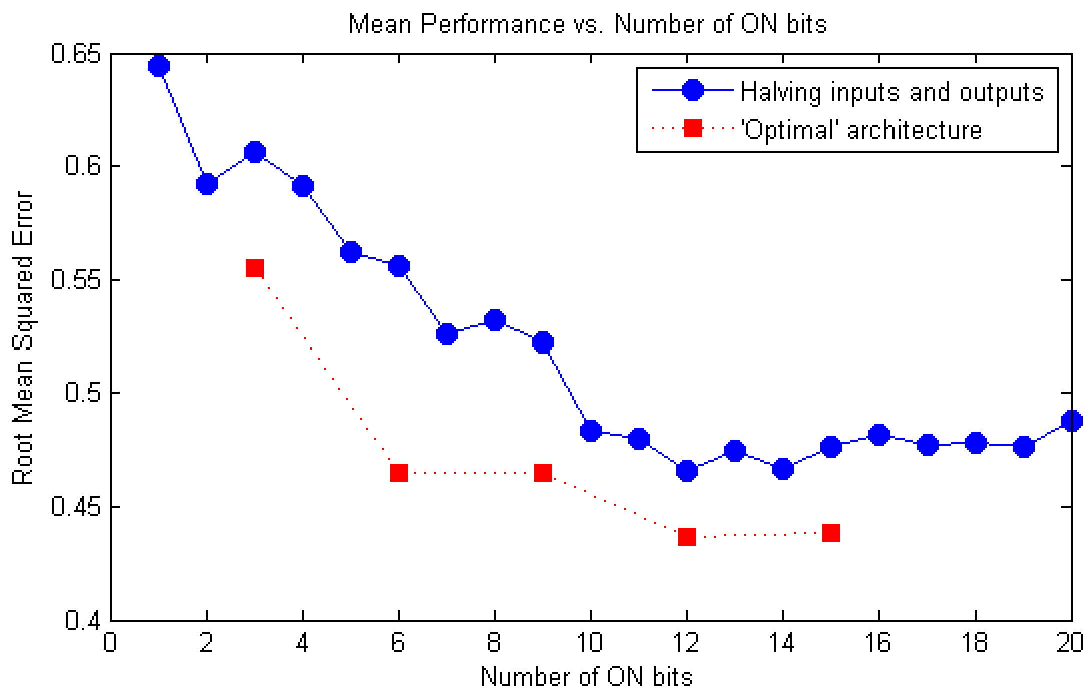

Figure 5.8: Performance difference between the 'optimal' architectures and the estimated architectures

As illustrated in Figure 5.8, the performance of the 'optimal' architectures has significant advantages over the estimated architectures in SGNO.

5.7 Symbolic Regression using Genetic Programming

In this application, MBE-GP is employed to discover concise relationships between the variables explored in SGNO and the longitudinal dispersion coefficient. MBE-GP symbolic regression is carried out at several key points, where various numbers (3, 6, 9 and 12) of variables are selected out of the total 20 input variables. For the case of 3 input variables, the population size is estimated at 105 using the rule of thumbs (Equation 3.10), the maximum number of expression tree branches in a chromosome is 5, the initial maximum tree depth is 3, and the size of the elite group is estimated at 5 (5% of the total population). Figure 5.9 illustrates the progress of the MBE-GP in terms of RMSE for the case of 3 variables selected from the SGNO rank table.

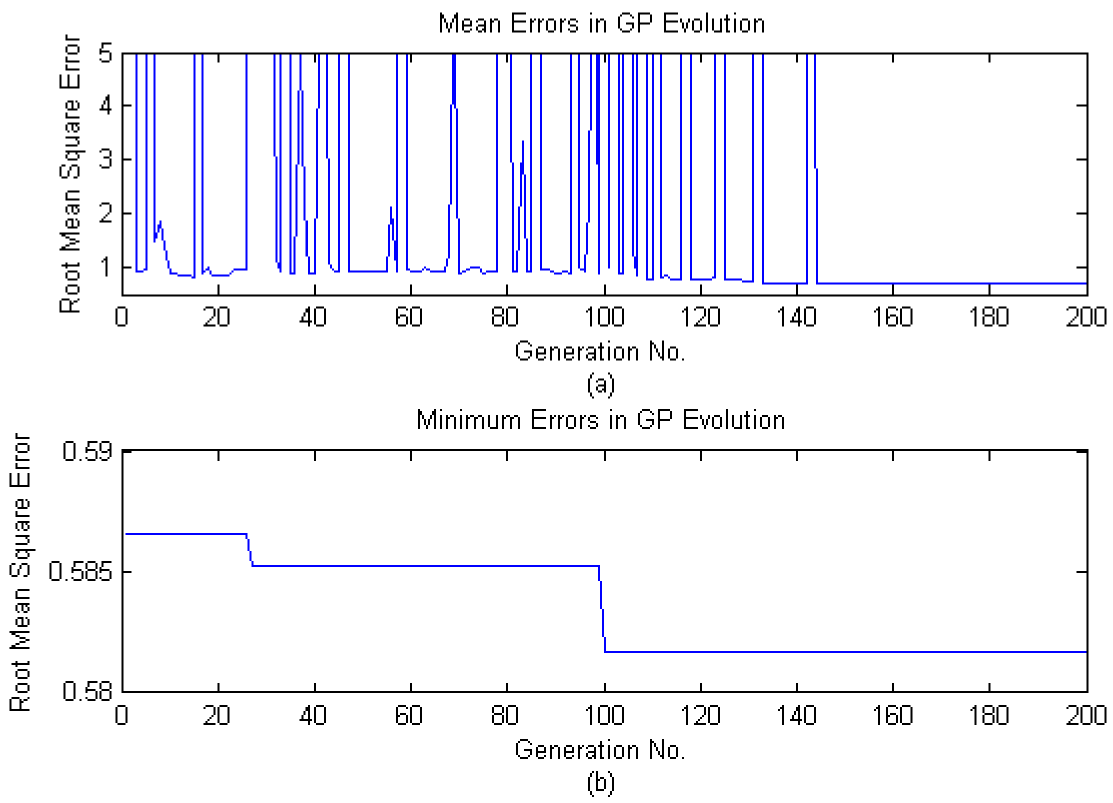

Figure 5.9: Evolution of MBE-GP for 3 variables. (a) mean errors in generations, (b) minimum errors in generations

As shown in Figure 5.9(a), the mean errors for the generations gradually decrease though there are many generations that generate extraordinarily large mean errors due to the appearances of certain chromosomes that produce very large regression errors. Figure 5.9(b) illustrates the minimum error in each generation and 3 decreasing steps are clearly shown with the global minimum RMSE, 0.5817, produced at the 100th generation. The chromosome with the minimum RMSE can be expressed using Equation 5.5. Comparing with the NN models with estimated architecture, which produce RMSE at 0.6060, this chromosome performs better.

However, this chromosome does not perform as well as the ‘optimal’ NN models, which produces RMSE at 0.5547.

$$\boldsymbol{y = A * x_1 + B * (x_2 - x_3)^3 + C * x_3 + D * x_1 x_2 + E} \tag{5.5}$$

Table 5.7 lists the performance of the GP process and the discovered symbolic expressions using 6, 9 and 12 variables from the SGNO rank table. Comparing with the NN models using the same input variables, these chromosomes generally perform better than the NN models with estimated architectures but not as well as those with ‘optimal’ architectures.

Table 5.7: Symbolic expressions discovered by GP for various numbers of variables

No. of Variables	RMSE	Symbolic Expression
6	0.5425	$y = A * x_5 + B * x_6^4 + C * (x_2 x_4 + x_1) + D * x_1 + E$
9	0.5201	$y = A * x_1 + B * x_7^{-4} + C * (x_5 + x_3 - x_1) + D * (x_1 x_2 x_9) + E * x_2 + F$
12	0.5160	$y = A * x_1 + B * (x_3 x_5 + x_4) + C * \left(x_2 \frac{x_6^4}{x_5/x_4} + x_7 x_8 + (x_2 - x_{10}) \right) + D * x_{12} + E * x_2 + F$

5.8 Benchmarking with other Feature Selection (FS) Techniques

In this section, the performance of the SGNO in this application is compared with several other FS techniques, including PCA, BFS, FFS and GNMM by evaluating the variable rank tables explored by all these techniques.

5.8.1 Principal Component Analysis (PCA)

PCA is one of the most famous mathematical techniques widely used in data dimensionality reduction (Dunteman, 1989). The details of PCA were discussed in section 3.3.1. In this application, PCA is used to transform the 20 original variables, excluding the longitudinal dispersion coefficients, into a new data space and the newly generated variables are then used to estimate the longitudinal dispersion coefficient using NNs. Figure 5.10 illustrates the variances carried in each PC. As shown in Figure 5.10, only 9 PCs are present in the figure. The first PC carries over 50% of the variance in the original dataset, significantly greater than the other PCs. The accumulated variance plot shows that the first 4 PCs explain about 80% of the variance and the first 6 PCs explain around 90%.

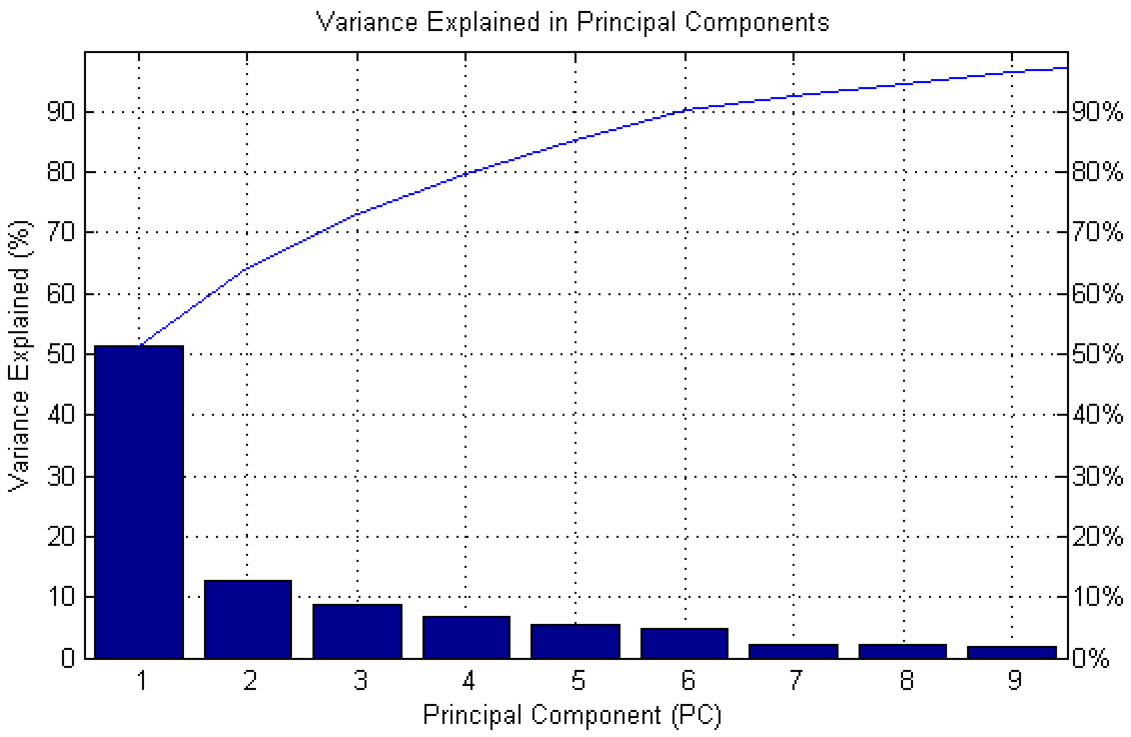

Figure 5.10: Variance accounted for or explained in PCs

Following the PCA transformation, the NN models are used to perform the longitudinal dispersion coefficients estimation using various numbers of PCs as the input variables. Five-fold cross-validation and early-stopping training is used in the NN models. Table 5.8 lists the performance of the NN models using different numbers of PCs. Figure 5.11 demonstrates the performance of these NN models against the NN models using the same numbers of variables selected from the SGNO's importance rank table.

Table 5.8: PCA re-evaluation results

Size	1	2	3	4	5	6	7	8	9
RMSE	0.6439	0.5918	0.606	0.591	0.5618	0.5559	0.5256	0.5317	0.5221

Size	10	11	12	13	14	15	16	17	18
RMSE	0.4835	0.4798	0.4654	0.4744	0.467	0.4764	0.4817	0.4771	0.4782

Size	19	20
RMSE	0.4767	0.4879

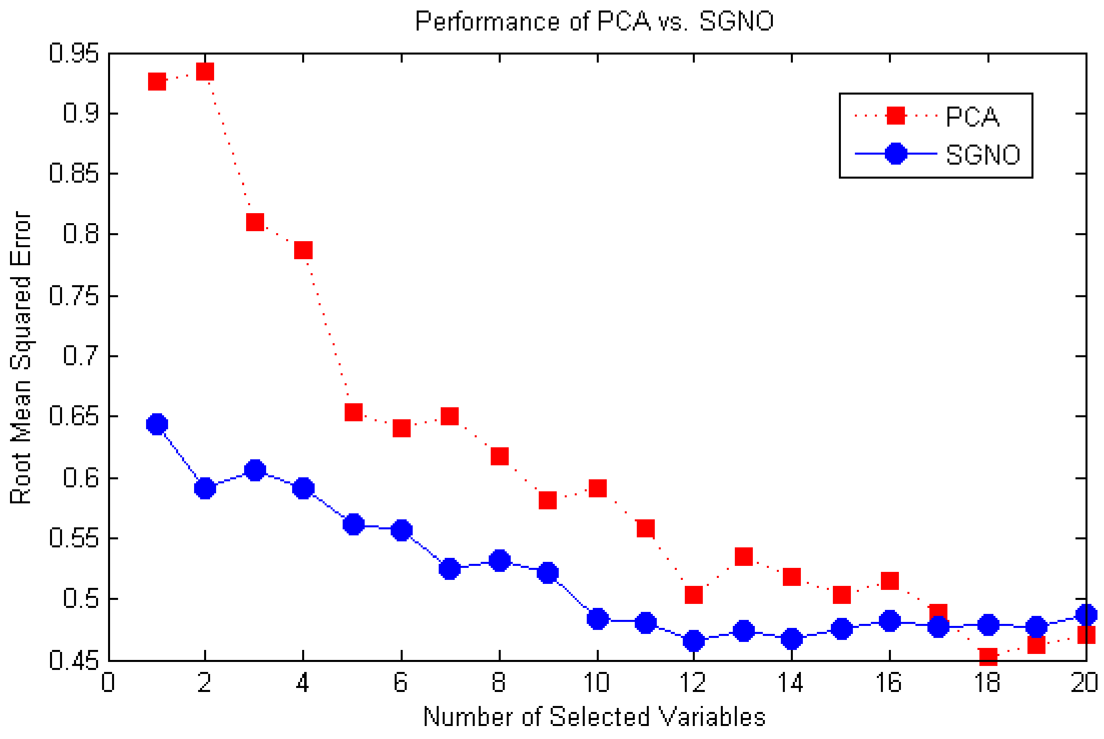

Figure 5.11: Performance of NN models using PCA discovery against SGNO

As shown in Figure 5.11, SGNO has a distinct advantage over the PCA as the RMSEs produced using the SGNO rank table are always less than those for PCA, especially for the cases where the selected number of variables is small. The largest difference is up to the value of 0.3420 (2 input variables). The advantage of SGNO reduces as the number of selected variables increases. Although PCA reduces the data dimensionality while preserving large amounts of variations in the original data, using a small number of PCs to predict the dispersion coefficient cannot achieve the same level of accuracy as the NN models taking the same number of input variables selected from the SGNO's rank table.

5.8.2 Forward Feature Selection (FFS) and Backward Feature Selection (BFS)

FFS and BFS are two commonly used sequential FS techniques. The detailed procedures of FFS and BFS are introduced in section 3.3.2. In this application, FFS and BFS employ NN models to evaluate the performance/efficiency of the selected variables during the process. The NN models employ the same architectures as the NN module in the SGNO process, i.e. single hidden layer with the number of hidden neurons estimated by halving the number of inputs and outputs and the transfer functions are tangent sigmoid and pure linear in hidden layer and output layer respectively. The following lists present the order in which variables are added to the selection list in FFS and the reversed order of variables removed from the selection in BFS. Hence, the variables are in descending order of preferences in both lists.

FFS: [20, 18, 4, 12, 1, 8, 7, 9, 16, 3, 6, 17, 10, 19, 14, 11, 15, 5, 2, 13]

BFS: [20, 9, 10, 12, 19, 18, 17, 16, 4, 15, 3, 1, 14, 2, 6, 8, 7, 11, 5, 13]

Comparing with the preference list (influence rank table) discovered using SGNO, which is **[20, 16, 10, 15, 6, 18, 2, 19, 9, 12, 1, 17, 7, 3, 11, 4, 14, 5, 8, 13]**, they all identify the 20th variable, tracer mean velocity, as the most important variable and the 13th variable, reach slope, as the least significant variable. The variables lying in between are of different orders in these lists. Table 5.9 lists the performance of the preference lists produced by BFS and FFS using NN models with various numbers of input variables selected from the preference lists. Figure 5.12 illustrates the performance of BFS and FFS compared with SGNO.

Table 5.9: BFS and FFS re-evaluation results

Size	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
BFS	0.6532	0.6406	0.6268	0.608	0.5871	0.5679	0.5396	0.5283	0.5009	0.4998	0.4921	0.4854	0.4708	0.4836	0.4622	0.474	0.4808	0.477	0.4847	0.4907
FFS	0.638	0.6085	0.6016	0.6086	0.5966	0.5764	0.552	0.5367	0.5064	0.507	0.4985	0.5092	0.4891	0.4854	0.4608	0.484	0.4896	0.4926	0.4859	0.4981

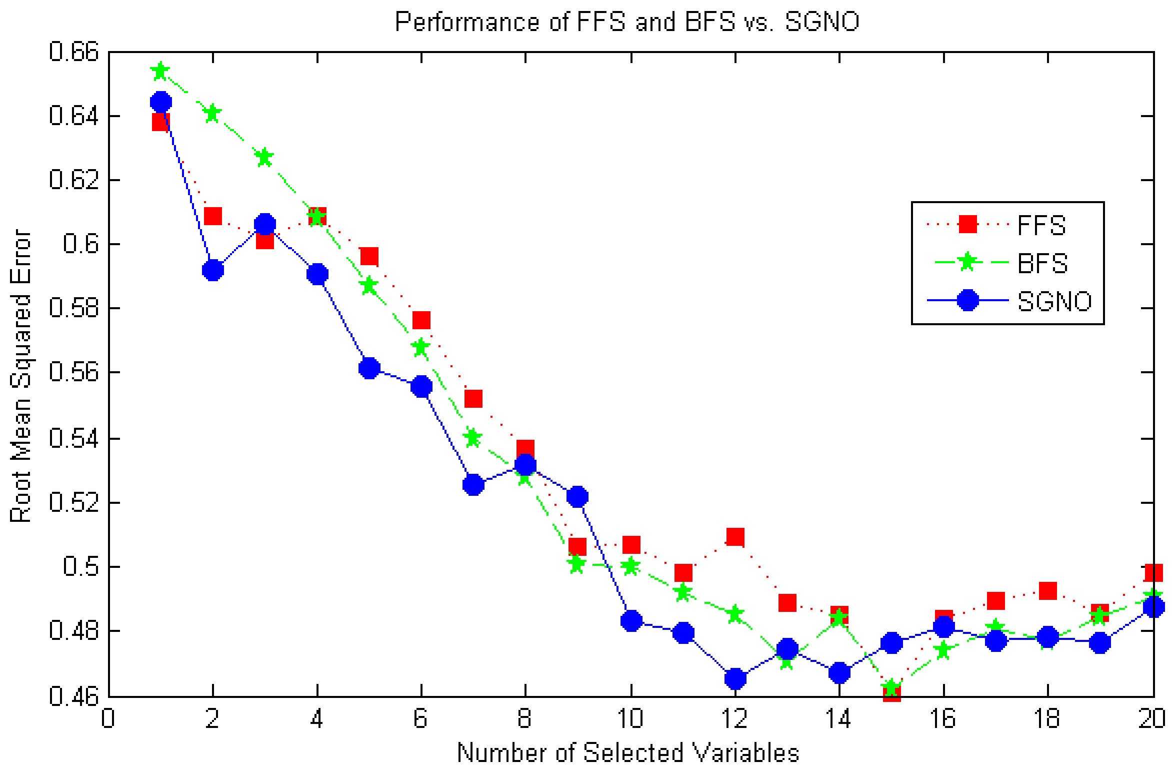

Figure 5.12: Performance of NN models using BFS and FFS against SGNO

As shown in Figure 5.12, FFS and BFS produce competitive performance against SGNO and the RMSE plots generated using these 3 techniques all follow a similar pattern, which gradually decreases as the number of selected variables increases. Although the RMSEs of these 3 techniques are within small ranges, SGNO produces smaller RMSEs compared with BFS and FFS at many points, especially when the number of selected variables is around 5 and 12. The possible reason is that BFS and FFS are considered to be 'local search' techniques, which search for improvement locally around the current state. In this application, BFS and FFS scan through all available variables, identify the one that generates the smallest error and updates the current state with that variable. On another hand, SGNO is a 'global' optimisation technique as it employs GA, a well known 'global search' algorithm with the ability to leave local minima, though the optimal solution is not guaranteed.

5.8.3 Genetic Neural Mathematical Method (GNMM)

As described in section 3.3.3, GNMM is a hybrid data mining technique incorporating GA and NNs. As GNMM employs GA, it is considered to be a global search algorithm as in the case of SGNO. In GNMM, variables are ordered in terms of their appearance frequencies/percentage during the evaluation of the GA process. The variables of higher appearance percentages are considered to be more favourable than those of lower appearance percentages (Yang, 2010; Yang et al., 2007). Figure 5.13 illustrates the appearance percentages of the variables discovered by the GNMM process.

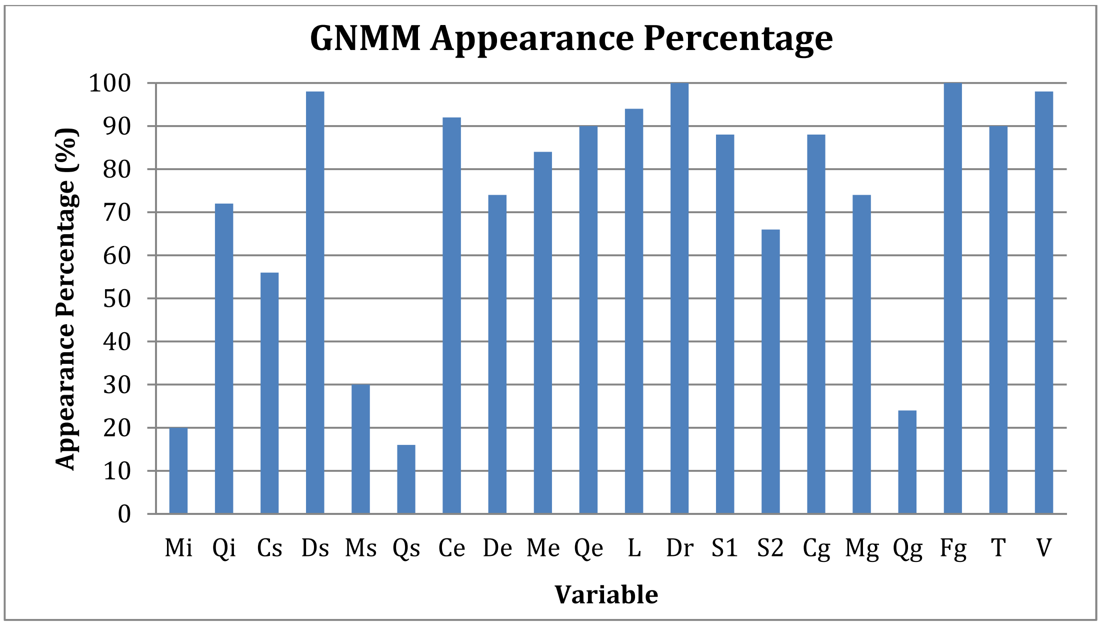

Figure 5.13: Variable appearance percentages in GNMM

As shown in Figure 5.13, the instantaneous flow rate measured at the gauging station, F_g, and the straight reach distance, D_r, are of the highest appearance percentages, 100%, followed by the distance from the injection point measured at the start location, D_s, and the mean velocity of the tracer, V. Most of the variables appear more than 50% and there are a total of 8 variables with appearance percentages no less than 90% during the GNMM process. The following list contains the ordered variables based on their appearance percentages in descending order.

[D_r, F_g, D_s, V, L, C_e, Q_e, T, S_1, C_g, M_e, D_e, M_g, Q_i, S_2, C_s, M_s, Q_g, M_i, Q_s]

The following index list contains the indices of the variables in the original dataset presented in the same order as the previous list.

[12, 18, 4, 20, 11, 7, 10, 19, 13, 15, 9, 8, 16, 2, 14, 3, 5, 17, 1, 6]

Comparing with the variable rank lists produced using SGNO, BFS and FFS in section 5.8.2, the variable rank list generated by GNMM is distinct from the others. However, GNMM identifies 2 variables with indices 18 and 20, which are of high preferences in other rank lists. Table 5.10 lists the performance of the rank list produced by GNMM using NN models with various numbers of input variables selected from the lists. Figure 5.14 illustrates the performance of rank list against SGNO.

Table 5.10: GNMM re-evaluation results

Size	1	2	3	4	5	6	7	8	9
RMSE	0.9288	0.7774	0.7833	0.6222	0.6027	0.5882	0.5804	0.5677	0.5923

Size	10	11	12	13	14	15	16	17	18
RMSE	0.5881	0.5451	0.5462	0.5266	0.5142	0.5084	0.5056	0.5	0.492

Size	19	20
RMSE	0.4887	0.4917

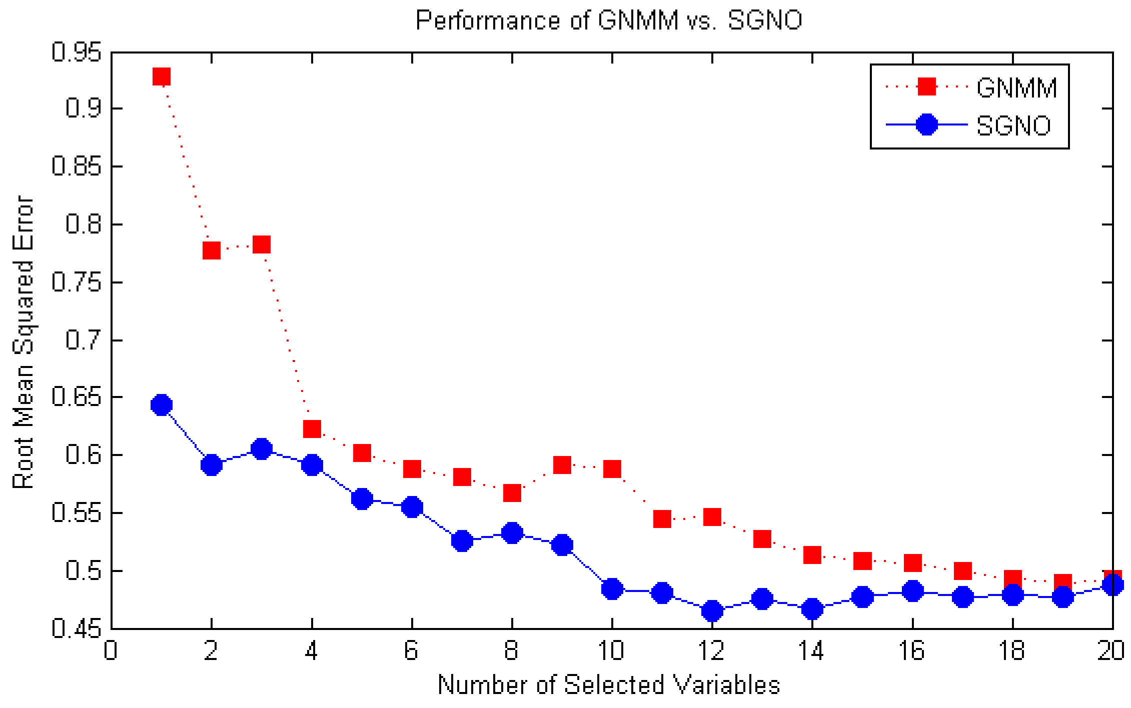

Figure 5.14: Performance of GNMM against SGNO

As shown in figure 5.14 that the variable rank list discovered by SGNO is of clear advantage over the one discovered by GNMM as the SGNO plot is always lower than the GNMM plot. The maximum difference between the RMSEs generated by these two techniques, which is 0.2849, occurs when only one input variable is used in the NN models. The difference dramatically decreases at the beginnings of the plots for up to 4 variables; these indicate that the first 3 variables are optimal or close. Following the dramatic drop, the GNMM plot gradually decreases with variations in the middle of the plot. At the end of the GNMM plot, RMSE keeps decreasing and approaching the RMSE plot of SGNO.

5.8.4 Benchmarking summary

Figure 5.15 summarises the RMSEs of the rank lists produced using the all the techniques presented in this chapter. It shows that these techniques produce similar results when the number of selected variables is large, say greater than 12. The key performance differences between these techniques occurs at the early stages of the RMSE plots, where PCA and GNMM produce the highest RMSEs and BFS, FFS and SGNO generate significantly better results. Among these 5 techniques, SGNO produces the best performance in general.

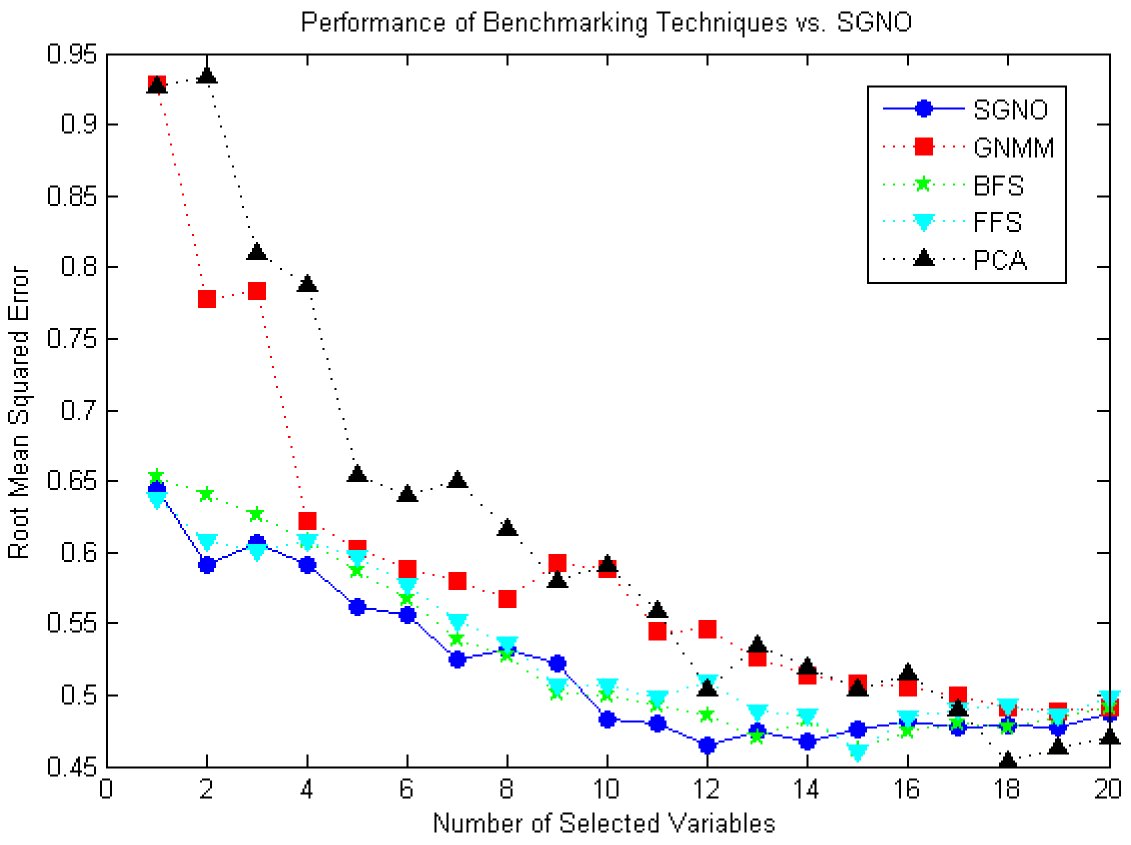

Figure 5.15: Performance of all benchmarking techniques against SGNO

5.9 Conclusion

In this chapter, SGNO is applied to establish a model to estimate a fundamental hydrodynamic characteristic coefficient, the longitudinal dispersion coefficient, using a dataset created by the UK Environment Agency (EA). The dataset contains geographical, geometrical and hydraulic parameters collected at 27 different rivers in the EA region. Apart from establishing prediction models, the symbolic relationship between the model inputs and output was extracted using multiple-branch GP.

Analysis of the results show that most of the variables contribute to the prediction of the longitudinal dispersion coefficient and increasing the number of inputs variables can generally increase the accuracy of the prediction until a certain stage where adding more variables does not improve the prediction accuracy (see Figures 5.7 and 5.15). The models established during the SGNO process are capable of producing errors within the range (0.465, 0.644) depending on the number of input variables using the architectures employed by the NN module in SGNO (see Figure 5.7). As the NN architectures used in the NN module are estimated using rules of thumb, employing 'optimal' architectures can improve the prediction accuracy significantly (see Figure 5.8).

After identifying the influential levels of all the variables, GP with tree structured multiple branches is applied to extract concise symbolic relationship between various numbers of input variables and the longitudinal dispersion coefficient, and the relationships discovered by GP are competitive to the prediction models established using NNs.

Finally, 4 benchmarking techniques, including BFS, FFS, GNMM and PCA, are implemented to evaluate the performance of SGNO by comparing the effectiveness of the variable rank tables discovered using all these 5 techniques. Among these 4 benchmarking techniques, PCA generates the highest RMSEs; GNMM produce similar trend as PCA but with lower RMSEs; BFS and FFS generate RMSEs of small differences and the RMSEs are generally lower than those produced by PCA and GNMM, especially in the cases of small numbers of input variables (see Figure 5.15). Comparing with these benchmarking techniques, SGNO is of significant advantage over PCA and GNMM. Although SGNO generates similar RMSEs as BFS and FFS, it shows a general advantage when compared with these 2 techniques.

References

Ahmed F.E. (2005) Artificial neural networks for disgnosis and survival prediction in colon cancer. Molecular Cancer 4.

Ahsan N. (2008) Estimating the Coefficient of Dispersion for a Natural Stream. World Academy of Science, Engineering and Technology 44:131-135.

Azamathulla H.M., Ghani A.A. (2010) Genetic Programming for Predicting Longitudinal Dispersion Coefficients in Streams. Water Resource Management 10.1007/s11269-010-9759-9.

Azamathulla H.M., Wu F.C. (2011) Support vector machine approach for longitudinal dispersion coefficients in natural streams. Applied Soft Computing 11:2902-2905.

Deng Z.Q., Bengtsson L., Singh V.P., Adrian D.D. (2002) Longitudinal Dispersion Coefficient in Single Channel Streams. Journal of Hydraulic Engineering 128:901-916.

Dunteman G.H. (1989) Principal Components Analysis SAGE Publications.

Elder J.W. (1959) The dispersion of a marked fluid in turbulent shear flow. Journal of Fluid Mechanics 5:544-560.

Fischer H.B. (1967) The Mechanics of Dispersion in Natural Streams. Journal of Hydraulics Division 93:187-216.

Fischer H.B., List E.J., Koh R.C.Y., Imberger J., Brooks N.H. (1979) Mixing in inland and coastal waters Academic Press, New York.

Guymer I. (1999) A National Dabatase of Travel Time, Dispersion and Methodologies for the Prediction of River Abstractions. Environment Agency R&D Technical Report P346.

Heaton J.T. (2005) Introduction to Neural Networks with Java Heaton Research.

Jain A.K., Zongker D. (1997) Feature Selection: evaluation, application and small sample performance. IEEE Transactions PAMI 19:153-158.

Jobson H.E. (1996) Prediction of Traveltime and Longitudinal Dispersion in Rivers and Streams, USGS Water Resources Investigations Report, USGS, Water Resources Division, Reston.

Karagiannopoulos M., Anyfantis D., Kotsiantis S.B., Pintelas P.E. (2007) Feature selection for regression problems, HERCMS'07, Athens University of Economics and Business.

Kashefipur S.M., Falconer A. (2002) Longitudinal dispersion coefficients in natural channels, The fifth international hydro informatics conference, Cardiff University. pp. 95-102.

Koussis A.D., Rodriguez-Mirasol J. (1998) Hydraulic estimation of dispersion coefficient for streams. Journal of Hydraulic Engineering 124:317-320.

Pudil P., Novovicova J., Somol P. (2002) Feature selection toolbox software package. Pattern Recognition Letters 23:487-492.

Rowinski P.M., Piotrowski A., Napiorkowski J.J. (2005) Are artificial neural netwrok techniques relevant for the estimation of longitudinal dispersion coefficient in rivers? Hydrological Sciences Journal 50:175-187.

Rub G., Kruse R. (2010) Feature selection for wheat yield prediction, in: M. Bramer, et al. (Eds.), Research and Development in Intelligent Systems XXVI, Springer-Verlag, London.

Sahay R.R. (2010) Prediction of longitudinal dispersion coefficients in natural rivers using artificial neural networks. Environmental Fluid Mechanics 10.1007/s10652-010-9175-y.

Seo I.W., Cheong T.S. (1998) Prediction longitudinal dispersion coefficient in natural streams. Journal of Hydraulic Engineering 124:25-32.

Seo I.W., Baek K.O. (2004) Estimation of the Longitudinal Dispersion Coefficient Using the Velocity Profile in Natural Streams. Journal of Hydraulic Engineering 130:227-236.

Tayfur G., Singh V.P. (2005) Predicting longitudinal dispersion coefficient in natrual streams by artificial neural network. Journal of Hydraulic Engineering 131:991-1000.

Taylor G.I. (1954) Dispersion of matter in turbulent flow through a pipe. Proceedings of The Royal Society 223:446-468.

Toprak Z.F., Cigizoglu H.K. (2008) Predicting longitudinal dispersion coefficient in natural streams by artificial intelligence methods. Hydrological Processes 22:4106-4129.

Toprak Z.F., Savcl M.E., Avcl C. (2004) Predicting longitudinal dispersion coefficient in natural streams using fuzzy-logic. Hydrology Science and Practice for the 21st Century I:379-388.

Veliskova Y., Sokac M., Dulovicova R. (2009) Determination of Longitudinal Dispersion Coefficient in Sewer Networks, International Symposium on Water Management and Hydraulic Engineering, Ohrid/Macedonia. pp. 493-498.

Yang J.H. (2010) Intelligent data mining using artificial neural networks and genetic algorithms: techniques and applications, Engineering, University of Warwick, Coventry. pp. 260.

Yang J.H., Hines E.L., Iliescu D.D., Leeson M.S. (2007) GNMM and accurate longitudinal dispersion coefficient prediction, The Seventh UK Chinese

Association of Resource and Environment (CARE) Annual Meeting, Greenwich, London.

Zhang Y.X., Li H., Hou A.X., Havel J. (2006) Artificial neural networks based on principal component analysis input selection for quantification in overlapped capillary electrophoresis peaks. Chemometrics and intelligent laboratory systems 82:165-175.

CHAPTER 6

Wave Overtopping Prediction at Coastal Structures

6.1 Overview

In the previous chapter, SGNO was used to discover the relationship between the longitudinal dispersion coefficient and several hydraulic and geometrical variables. In this chapter, SGNO is applied to a data set selected from the project 'CLASH' (Crest Level Assessment of coastal Structures by full scale monitoring, neural network prediction and Hazard analysis on permissible wave overtopping) to study the relationship between wave overtopping discharges and several environmental and geometrical measurements.

Seawalls defend land and properties at the coastal regions against storm waves and sea wave run-ups. Once the highest run-up level exceeds the crest, a certain amount of sea water will be transported over the crest of the coastal structure and the associated discharge may cause a hazard to the land and properties behind. This is referred to as 'wave overtopping' and the amount of sea water overtopping is known as 'overtopping discharge'. The structure of the seawalls plays a predominant role in preventing wave overtopping and controlling the overtopping discharge to an acceptable level is a critical factor in coastal structure design (Goda, 2009; Ingram et al., 2009).

To assist with the design of coastal structures, accurate wave overtopping estimation models are required to evaluate the effectiveness of prototypes. However, there is lack of reliable and robust prediction methods for wave overtopping as wave overtopping is a complicated natural phenomenon depending not only on environmental conditions, such as wave height, wave period, water level, etc., but also depending on the geometrical structure and the properties of the building materials of the coastal structure (Soliman, 2003). The most frequently used methods are empirical models which are usually built based on laboratory experiments. These empirical models can only be applied with severe restrictions due to restricted test data ranges and structure configurations (Soliman, 2003; Verhaeghe, 2005).

6.2 Background

Wave overtopping at coastal structures have been continuously studied for several decades by researchers all over the world. Wave overtopping discharges may vary up to several orders of magnitude under random wave attacks. In general, wave overtopping discharge is defined as the volume of water exceeding crest level per unit time and unit structure width, i.e. $m^3s^{-1}m^{-1}$. Wave overtopping cannot be avoided completely due to the random nature of waves (especially under severe environmental conditions) and the trade-off between economical factors (costs, etc.) and environmental factors (Soliman, 2003). Hence, predicting overtopping of structures has become an important evaluation method to support the design of coastal structures.

Saville (1955) was one of the researchers who initiated studies on the overtopping of regular waves. Since then, several models have been developed to predict wave overtopping at various structure types. Overtopping models can be categorised into two distinct groups, which are empirical models and numerical models. Empirical models are the most studied and widely used models of wave overtopping. They are also known as regression models, which are based on the available overtopping data collected from laboratory experiments and field experiments. Numerical models analyse wave behaviours by solving a series of equations expressing fluid flow in front of the structure (Verhaeghe, 2005).

Owen (1980) proposed an overtopping model for a smooth sloping structure. The model suggests an exponential relationship between the dimensionless mean overtopping discharge, *Q*, and the dimensionless crest freeboard, *R*.

$$\boldsymbol{Q = a \cdot exp(-b \cdot R)} \tag{6.1}$$

$$\boldsymbol{Q = \frac{q}{gH_sT_m}} \tag{6.2}$$

$$\boldsymbol{R = \frac{R_c}{T_m\sqrt{gH_s}}} \tag{6.3}$$

where q is the mean overtopping discharge; *g* is gravity; H_s is the mean wave height; T_m is the mean period of the incident waves, R_c is the crest freeboard, and *a* and *b* are the regression coefficients determined based on these structure characteristics.

Following the study by Owen (1980), Ahrens, Heimbaugh and Davidson (1986) studied overtopping at various types of structures and proposed another exponential model, which can be expressed as follows:

$$\boldsymbol{Q = a \cdot exp(-b \cdot R)} \tag{6.4}$$

$$\boldsymbol{Q} = \frac{\boldsymbol{q}}{\sqrt{\boldsymbol{g}\boldsymbol{H}_s^3}} \tag{6.5}$$

$$\boldsymbol{R} = \frac{\boldsymbol{R}_c}{(\boldsymbol{H}_s^2\boldsymbol{L}_p)^{1/3}} \tag{6.6}$$

where a and *b* are the regression coefficients; *q* is the mean overtopping discharge; *g* is gravity; H_s is the mean wave height and L_p is the mean wave length.

In recent years, NNs have been successfully applied by many researchers in various fields of coastal engineering. The success of NNs is mainly based on their ability to approximate virtually any continuous non-linear functions (Cybenko, 1989; Verhaeghe, 2005). In 1995, Mase, Sakamoto and Sakai (1995) studied the applicability of NNs as a tool to model the stabilities of various rock slopes under random wave attacks. Their work showed that NNs are capable of predicting the damages levels of the structures with correlation coefficient up to 0.94 between the measured stability and the predicted stability using 7 parameters of wave hydraulics and structural properties. In 1999, Medina (1999) proposed 2 NN models predicting regular wave run-ups at a conventional rubble mound breakwater and a dissipating basin breakwater respectively. Evolutionary Algorithms (EAs) were used to optimise the topologies of these 2 NN models. The NN model for the conventional breakwater is of reasonable accuracy, while the model for the dissipating basin breakwater is of poorer accuracy. Years later, Medina, Gonzalez-Escriva and De Rouck (2002) developed a chain of 2 NN models to predict the occurrence of significant overtopping and the amount of overtopping discharge respectively. The NN models are simple 1 hidden layer MLPs with 4 input parameters. Their results showed that these 2 NN models are efficient and perform better than an empirical model.

Since the launch of the European CLASH project, many researchers have published their attempts at the modelling of wave overtopping using the data sets collected in the project. In 2005, Van Gent, Pozueta, Van den Boogaard and Medina (2005) presented their NN model to predict wave overtopping discharge in the final CLASH work package report. The proposed NN model is a single-hidden-layer MLP with 20 hidden processing neurons and 15 input parameters covering hydraulic and structural properties. The result demonstrated that the NN model can successfully model the relationship between the input parameters and the overtopping discharge. Verhaeghe (2005, 2008) developed a 2-phase NN model consisting of a classifier, which aims to detect the occurrence of significant overtopping, and a quantifier, which is used to predict the amount of overtopping discharge if it occurs. The proposed 2-phase NN model is similar to the NN model chain presented by Medina et al (2002) but more complicated as it takes 14 input parameters instead of the 4 in Medina's model. The classifier and the quantifier are both simple MLPs of a single hidden processing layer; the classifier has 20 hidden neurons while the quantifier has 25 hidden neurons. The results show that the classifier can achieve up to 81% classification accuracy and the quantifier can predict the overtopping discharge with an RMSE as low as 0.52.

6.3 Data set

The data set used in this study is the database created in the European project 'CLASH', which ran from January 2002 until December 2004 (Verhaeghe, 2005). The CLASH project aimed to improve the knowledge on the phenomenon of wave overtopping and develop generic prediction methods based on many existing data sets on wave overtopping (De Rouck et al., 2008; Verhaeghe, 2005). Based on this dataset, Verhaeghe (2005) developed a 2-phase classifier-quantifier model, in which the classifier identifies if significant overtopping is likely to happen, and the quantifier estimates the amount of overtopping if significant overtopping is identified. Compared with Verhaeghe's model, SGNO establishes overtopping quantifier models with less number of inputs.

The CLASH database gathered over 10000 overtopping test entries carried out at various institutes and sites all over the world during the past few decades. Each of these tests is described by a total of 30 parameters representing hydraulic (i.e. measured overtopping discharge, wave period, wave height, etc.), structural (i.e. crest height, slope angle, building material, etc.) and general (i.e. test ID, reliability factor and complexity factor) characteristics (De Rouck et al., 2008; Verhaeghe, 2005). The reliability of each test in the database is estimated using the reliability factor, which is an integer in the range from 1 to 4. A reliability factor assigned to 1 indicates the corresponding test is very reliable, while the value 4 indicates that the test is not reliable. In the same way, the complexity of coastal structure in each test is indicated using an integer complexity factor in the range from 1 to 4, where 1 indicates a very simple structure and 4 indicates a very complex structure (van Gent et al., 2005; van Gent et al., 2007; Verhaeghe, 2005; Verhaeghe et al., 2008).

As the CLASH database is a collection of overtopping tests originated from over 100 independent tests carried out at various institutes and fields, the collected parameters in each test may not be the same and several parameters were not available in some tests (Verhaeghe, 2005). Hence, Verhaeghe (2005) suggested that only a set of 16 characteristic parameters among all the available parameters should be used to build the NN prediction model. The selected parameters are listed in Table 6.1. Figure 6.1 illustrates the parameters with a simple coastal structure.

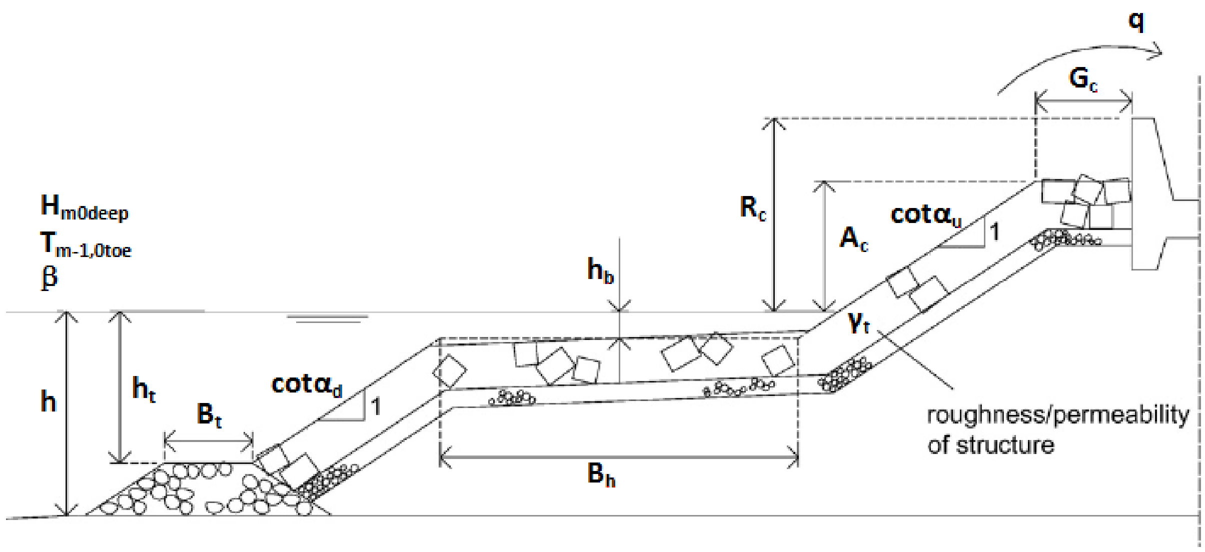

Figure 6.1: Parameters explained (adopted from Verhaeghe, 2005)

Table 6.1: Parameters suggested by Verhaeghe (2005)

Nature	Parameter	Unit	Description
Hydraulic	q	$m^3s^{-1}m^{-1}$	Mean overtopping discharge
	H_{m0deep}	m	Wave height determined at deep water
	H_{m0toe}	m	Wave height determined at the toe of the structure
	$T_{m-1,0toe}$	s	Mean wave period at the toe of the structure
	β	o	Angle of the incident wave
Structural	h	m	Water depth in front of the structure
	h_t	m	Water depth on the toe of the structure

	B_t	m	Width of the toe of the structure
	γ_t	–	Roughness factor of the structure
	$cot\alpha_d$	–	Cotangent of the angle of the structure below the berm
	$cot\alpha_u$	–	Cotangent of the angle of the structure above the berm
	R_c	m	Crest freeboard of the structure
	h_b	m	Water depth on the berm
	B_h	m	Horizontal width of the berm
	A_c	m	Armour crest freeboard of the structure
	G_c	m	Crest width of the structure

Among these parameters, q, the overtopping discharge is the output parameter and all the other parameters are the inputs. In this work, only the reliable test results with relatively simple structures, e.g. the reliable factor and the complexity factor are less than or equal to 2, and the occurrences of overtopping, i.e. $q > 0$, are considered. Under these conditions, a total of 5277 tests are selected.

6.4 Data Pre-processing

As the CLASH database is a collection of tests carried out by different institutes, part of the data concerns prototype measurements at real sites and the remaining data are laboratory based experiments under various model scales (Verhaeghe, 2005). Instead of rescaling parameters using min-max normalisation or mean-standard deviation normalisation, Verhaeghe (2005) and van Gent (2005) suggested the idea of rescaling the parameters in individual tests against the H_{m0toe} in each test using the Froude scaling law, where H_{m0toe} is rescaled to 1m. Unlike other scaling methods, such as min-max scaling and mean-standard deviation scaling, which scale variables in datasets, Froude scaling law provides entry-wise scaling in datasets. Froude scaling selects one of the variables in the dataset as the unit variable, which will be scaled to 1 in all data entries, to produce scale factors for other variables. From the Froude scaling law, the following scaling factors are derived and listed in Table 6.2.

Table 6.2: Parameter rescaling using Froude scaling law

Parameters	Unit	Scale Factor
q	$m^3s^{-1}m^{-1}$	$1/(H_{m0toe})^{1.5}$
$T_{m-1,0toe}$	s	$1/(H_{m0toe})^{0.5}$
H_{m0deep}, H_{m0toe}, h, h_t, B_t, R_c, h_b, B_h, A_c, G_c	m	$1/H_{m0toe}$
β, γ_t, $cot\alpha_d$ $cot\alpha_u$	–	1

All the parameters are rescaled by multiplying by the corresponding scale factor and the rescaled parameters are marked with an apostrophe (') after the parameter name, e.g. H_{m0deep}', to distinguish from the original parameters (Verhaeghe, 2005). Among the selected 5277 tests, the suggested 16 parameters are in different ranges. Table 6.3 list the basic statistics of these parameters after Froude rescaling.

Table 6.3: Statistics of parameters after rescaling

Parameter	Min	Max	Mean	Standard Deviation
q'	1.8032×10^{-7}	0.4222	0.0085	0.0223
H_{m0deep}'	0.6957	5.0952	1.0867	0.2862
H_{m0toe}'	1	1	1	0
$T_{m-1,0toe}'$	2.8868	69.4620	4.8477	3.0176

β'	0	80	3.6646	11.3691
h'	1.0236	23.3333	4.0229	2.5460
h_t'	0.5238	23.3333	3.7391	2.5384
B_t'	0	19.0476	0.5027	1.1912
γ_t'	0.33	1	0.6969	0.2837
$cot\alpha_d'$	0	7	2.2109	1.3808
$cot\alpha_u'$	-5	8	2.0649	1.7485
R_c'	0	6.4219	1.4481	0.6478
h_b'	-1.2109	7.8947	0.0724	0.4657
B_h'	0	33.8983	0.7041	2.2570
A_c'	0	6.2393	1.3539	0.6409
G_c'	0	6.6625	0.7892	0.9416

Once all parameters are scaled to H_{m0toe}=*1m*, the parameter H_{m0toe}' becomes a constant, 1, in all tests. Therefore, there is no need to use H_{m0toe}' as a separate input parameter and it can be omitted in later processing. It is noticeable in Table 6.3 that the overtopping discharge, *q'*, in the scaled data set is in an extraordinary range, where the maximum value, 0.4222, is a million times greater than the minimum value, 1.8x10^{-7}. Furthermore, examining the data set in detail shows that a significant amount (over 20%) of overtopping discharge measurements is valued less than 0.0001. Figure 6.2 is a histogram of the overtopping discharges.

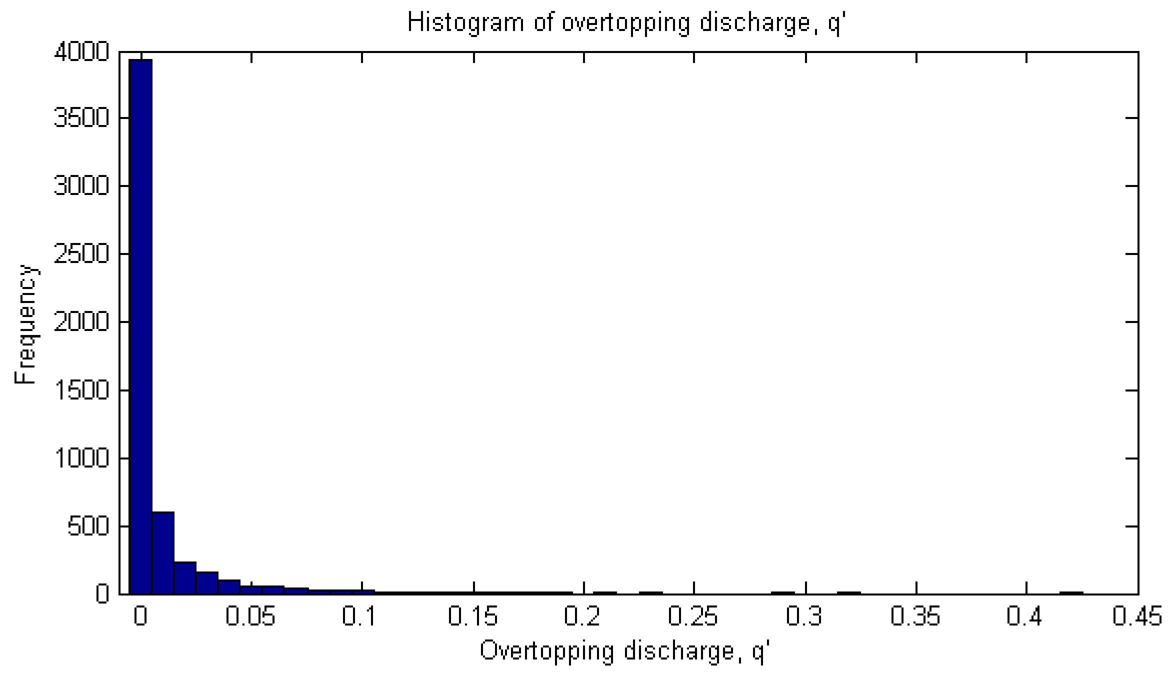

Figure 6.2: Histogram of overtopping discharges

Figure 6.2 shows clearly that distribution of overtopping discharges, q', is extremely imbalanced. Hence, to reduce the differences between measurements and to improve the distribution of the measurements, the logarithms of the overtopping discharges, *log(q')*, are used to construct the NN models instead of *q'* (van Gent et al., 2005; Verhaeghe, 2005; Verhaeghe et al., 2008). In addition, Verhaeghe et al (2008) showed that a NN trained with the *q'* only performs well for the largest *q*-values in the range between 0.01 and 1; a better result is obtained when an NN is trained with *log(q')*. The application of the logarithm can also avoid the occurrence of negative estimates of overtopping discharges. Figure 6.3 illustrates the histogram of the logarithms of overtopping discharges.

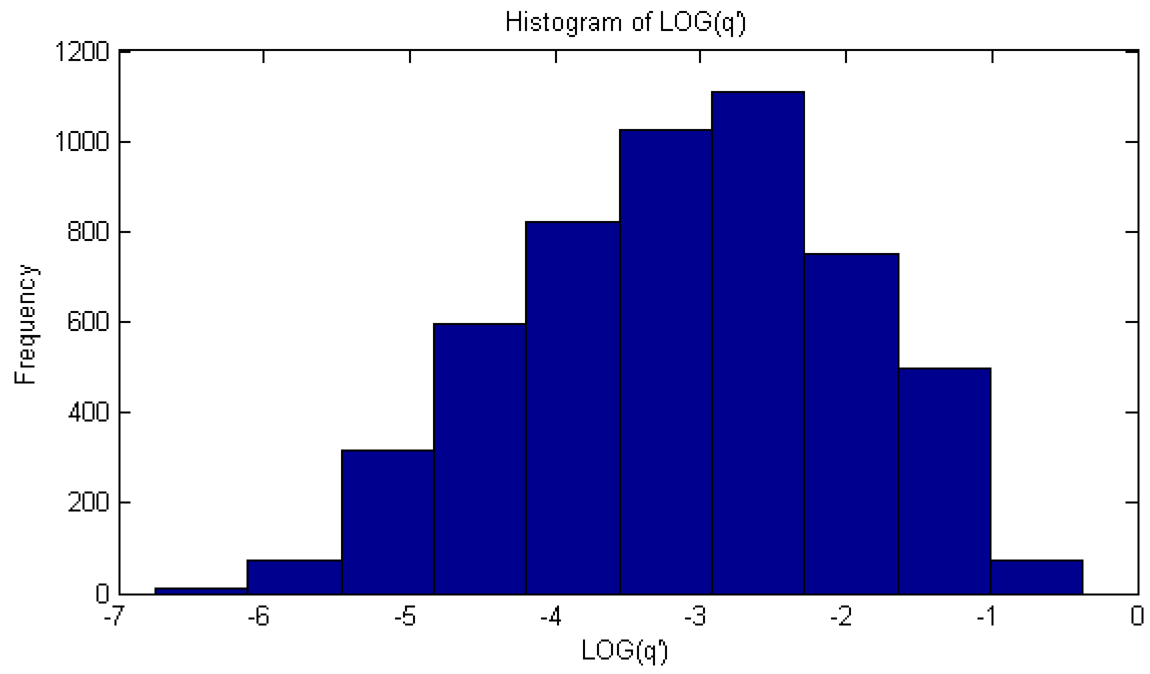

Figure 6.3: Histogram of the logarithm of overtopping discharges

The 5277 selected tests are divided randomly into five groups to apply the five-fold cross-validation (see section 3.2.2.3.2). Table 6.4 lists the distribution of the selected tests.

Table 6.4: Test distribution in five-fold cross-validation

	Set A	**Set B**	**Set C**	**Set D**	**Set E**	**Total**
Count	1055	1056	1055	1056	1055	5277

6.5 Variable Selection using Sensitive Genetic Neural Optimisation (SGNO)

6.5.1 Neural Network module

In this application, the NN module aims to establish the relationship between the parameters selected in the GA module and the logarithm of the corresponding overtopping discharge. As described previously in section 3.2.2.3, the NN module employs five-fold cross validation with early-stopping. The NNs have a single hidden processing layer and the number of hidden neurons in the processing layer is estimated by halving the total number of input and output variables of the NN. The initial weights in the hidden neurons are random numbers. The commonly used tangent-sigmoid function (see Equation 2.5) is used as the activation functions in the hidden layer and the pure linear function is used in the output layer (see Equation 2.2).

6.5.2 Genetic Algorithm module

In this application, the size of each binary chromosome in the GA module is 14 as there are 15 input parameters as suggested by Verhaeghe (2005) and the parameter H_{m0toe} can be removed after the Froude rescaling process. The size of the GA population is estimated at 70 using the rule of thumb discussed in section 3.2.2.2 (equation 3.3) and the size of the elite group is estimated at 3, which is about 5% of the total population. Figure 6.4 illustrates the performance of the chromosomes evaluated in 50 GA generations. It clearly shows that the mean RMSE in each generation drops rapidly in the first few generations followed by a steadily converged pattern. Figure 6.5 illustrates the size of chromosomes (number of ON bits) in the GA generations.

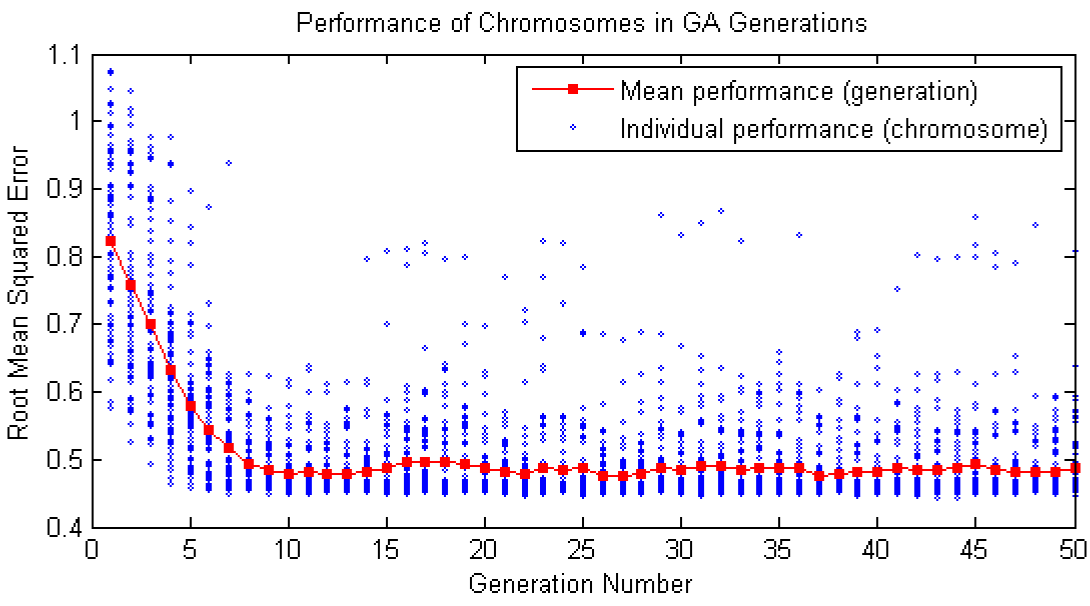

Figure 6.4: Chromosomes performance in GA generations

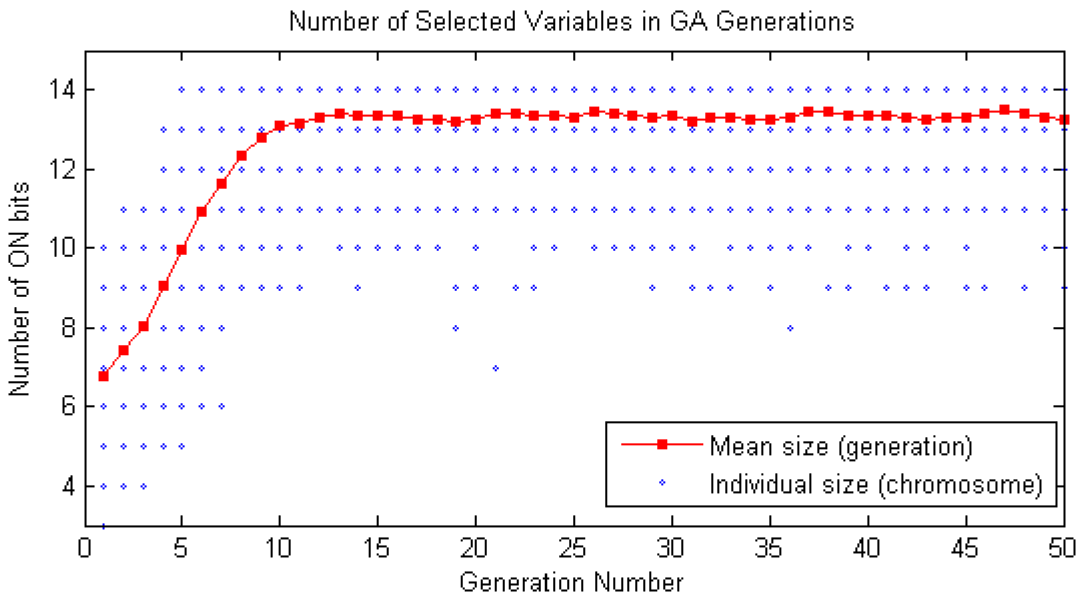

Figure 6.5: Number of ON bits in each generation

It is clear that Figure 6.5 shows a similar but inverse development trend as Figure 6.4, which illustrates the performance of the chromosomes. The mean size of chromosomes in each generation grows rapidly in the first few generations and then shows a converged pattern afterwards. In addition, the figure shows that the saturated chromosome, in which all 14 input parameters are selected, starts appearing at the 5th generation and remains throughout the entire GA process. The mean size of the chromosomes in each generation becomes over 13 from the 10th generation indicates that a significant amount of saturated chromosomes exists in each generation. Figure 6.6 illustrates the performance of chromosomes against their sizes. It is observable that the RMSEs of the chromosomes decrease as the number of ON bits (size of chromosomes) increases.

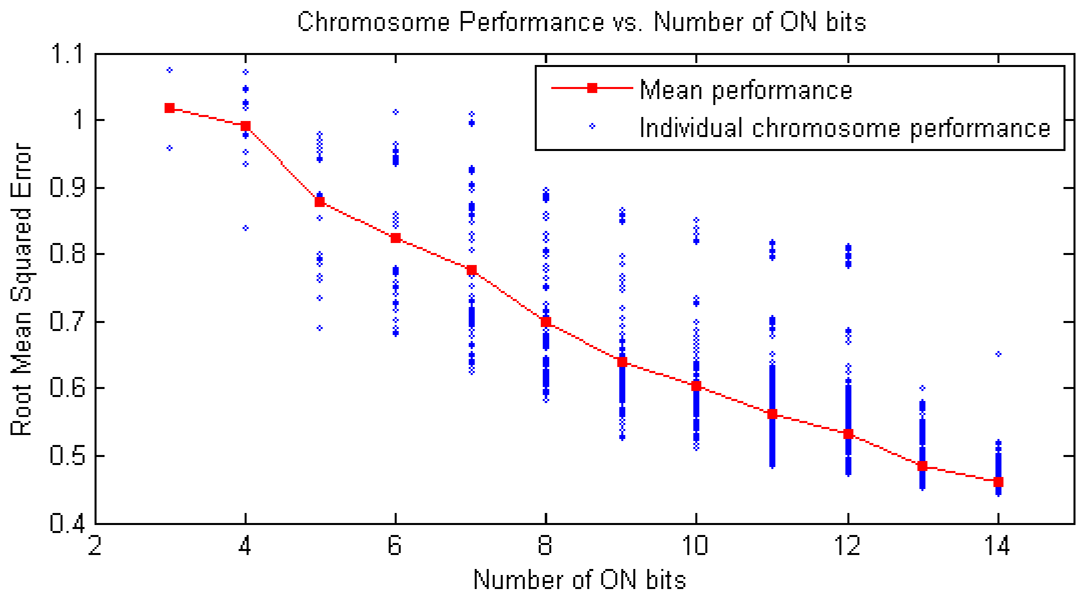

Figure 6.6: Chromosome performance vs. number of ON bits

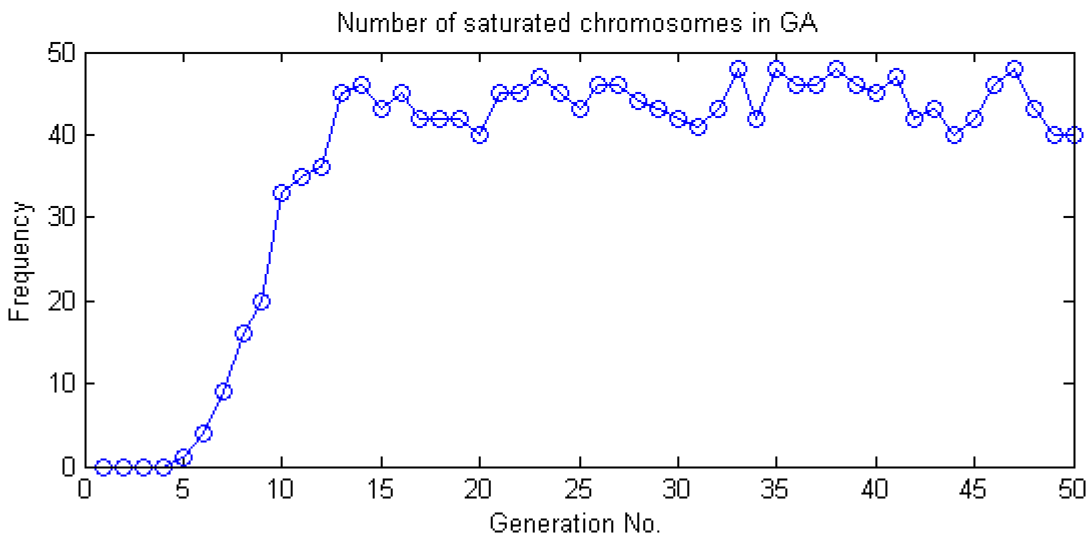

Figure 6.7 Number of saturated chromosomes in GA generations

Figure 6.7 illustrates the number of saturated chromosomes during the GA process. The number of saturated chromosomes keeps increasing starting at the 5th generation and fluctuates around 45, which is about 64% of the total population, since the 13th generation. Together with the patterns illustrated in Figure 6.4 and Figure 6.5, which tend to converge starting at the 10th generation, these indicate that the GA evolution process is only effective in the first 10 generations.

6.5.3 Sensitivity Analysis module

The SA module aims to identify the global influences of input parameters towards the output parameter by analysing the importance of input parameters appeared in the group of selected well-performing chromosomes in each generation.

As discussed in the previous section 6.5.2, the GA evolution process is effective only in the first 10 generations in this application. Hence, only the chromosomes evaluated in the first 10 generations are considered. In each generation, a quarter of the chromosomes with the highest performance in the population (70 x 25% ≈ 17) are selected to carry out the SA analysis. Figure 6.8 illustrates the size (number of ON bits) of the selected chromosomes. The figure shows that the saturated chromosomes still take a significant portion of the selected chromosome group (50 / 170 ≈ 29%).

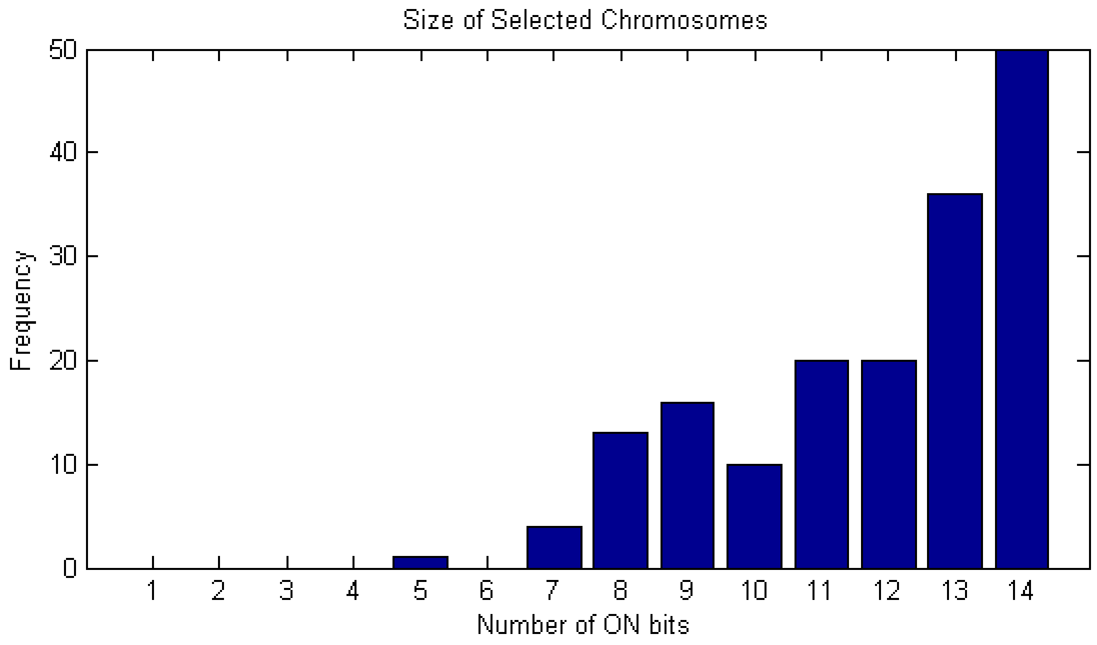

Figure 6.8: Sizes of the chromosomes selected for SA analysis

The global sensitivity score of a certain input parameter is calculated by taking the mean of the sensitivity scores derived in the selected chromosomes of high performance. Figure 6.9 illustrates the global sensitivity scores of all the input parameters. In the figure, the structure roughness, γ_t, wave height determined at deep water, H_{m0deep}, and the crest freeboard, R_c, have outstanding sensitivity scores comparing to other parameters. The angle of incident wave, β, has the lowest sensitivity score and hence the lowest influence on the model output.

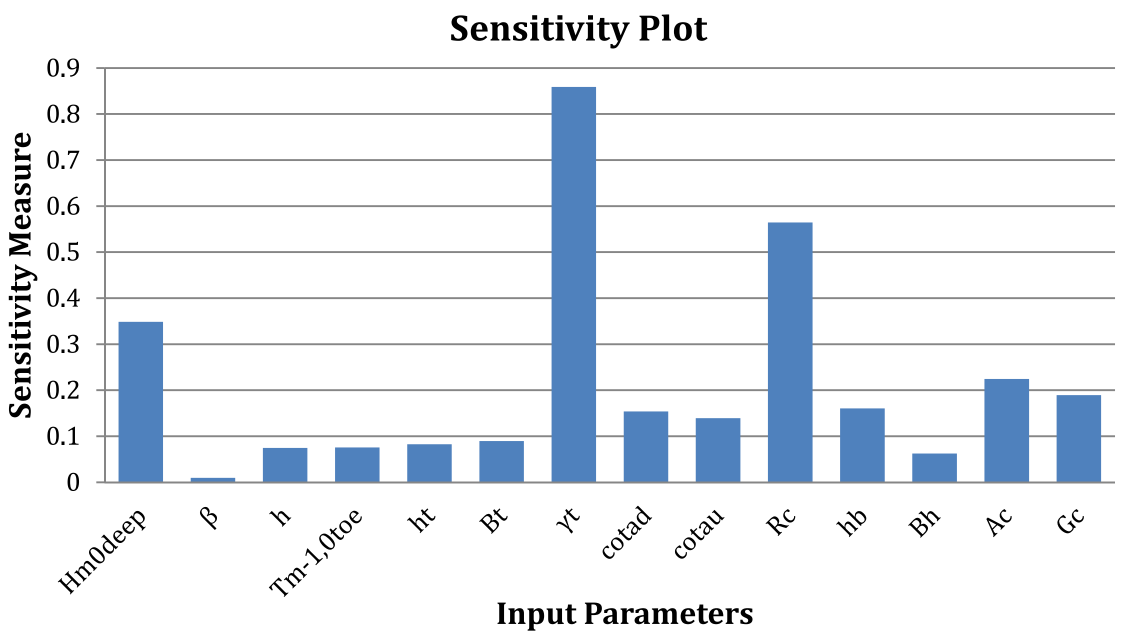

Figure 6.9: Global sensitivity scores of all input parameters

Based on the global sensitivity scores of the input parameters, the importance rank table can be constructed by rearranging the parameters in descending order. The parameter of higher sensitivity score is considered to be of higher importance/influence to the model output. The importance rank table is listed below, in which the former elements have higher sensitivity scores than the latter elements.

[γ_t R_c H_{m0deep} A_c G_c h_b $cota_d$ $cota_u$ B_t h_t $T_{m-1,0toe}$ h B_h β]

In terms of parameter indices in the chromosomes, the importance rank table can be re-expressed by replacing the input parameters with their positions in the chromosomes. Comparing to the parameter rank table listed above, the re-expressed index rank table is more understandable, especially for benchmarking in the later section 6.8. The indices list is

[7, 10, 1, 13, 14, 11, 8, 9, 6, 5, 4, 3, 12, 2]

6.6 System Remodelling

As mentioned in previous 3.2.3, NNs are not only used as the NN module in SGNO, but are also used to re-evaluate the performance of the ranked parameters in the importance rank table. Table 6.5 lists the performance of NN models using various numbers of input parameters selected from the SGNO's importance rank table. Figure 6.10 illustrates the results of NN models taking 3, 6 and 9 parameters from the rank table and figure 11 illustrates the performance of the NN models against the chromosomes evaluated in the GA module.

Table 6.5: SGNO re-evaluation results

Size	1	2	3	4	5	6	7	8	9
RMSE	1.006	0.8319	0.8152	0.78	0.7196	0.7178	0.6482	0.616	0.592
R^2	0.2113	0.4607	0.4821	0.5259	0.5965	0.5985	0.6726	0.7043	0.7269

Size	10	11	12	13	14
RMSE	0.5738	0.5141	0.5011	0.4613	0.4715
R^2	0.7434	0.7941	0.8043	0.8342	0.8267

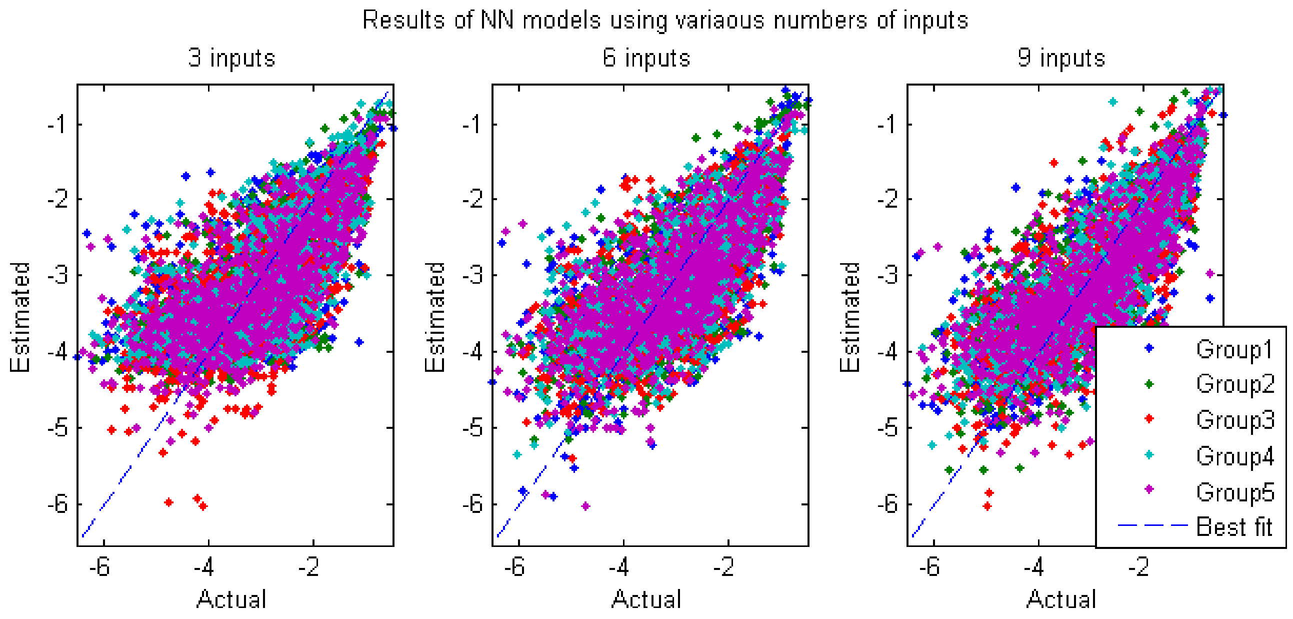

Figure 6.10: Performance of NN models taking 3, 6 and 9 inputs

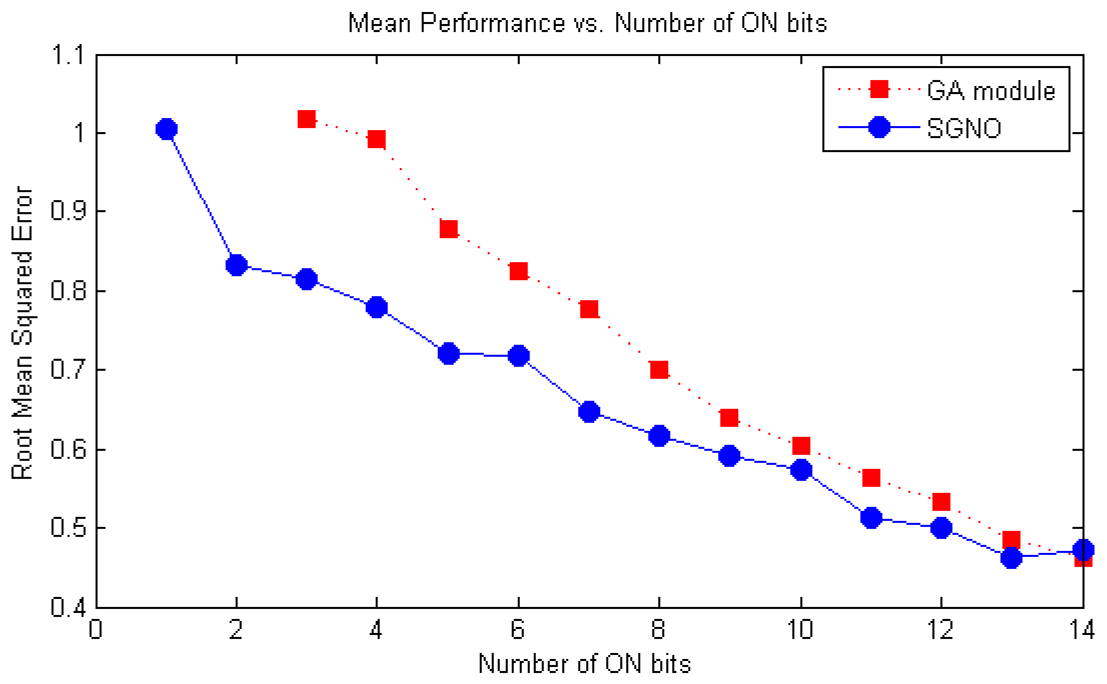

Figure 6.11: Performance of parameters in SGNO rank table vs. mean performance in GA module

Figure 6.11 clearly shows decreasing patterns for both plots, and the performance of the importance rank table constructed by SGNO is of significant advantage over the mean performance of the chromosomes evaluated during the GA process. As the number of parameters increases, the advantage of SGNO optimisation weakens and the difference between these two performance plots gradually decreases.

After evaluating the importance rank table, several NN models using various numbers of input variables are 'optimised' to discover the best performance achievable. Table 6.6 lists the 'optimal' NN architectures discovered and the best performance achieved for various numbers of input parameters selected from the importance rank table.

Table 6.6: 'Optimal' architectures and their performance

Number of input variables	'Optimal' architecture [1st layer, 2nd layer]	Performance (RMSE)
3	[6, 0]	0.7881
6	[11, 0]	0.6853
9	[11, 6]	0.5574
12	[15, 9]	0.4558

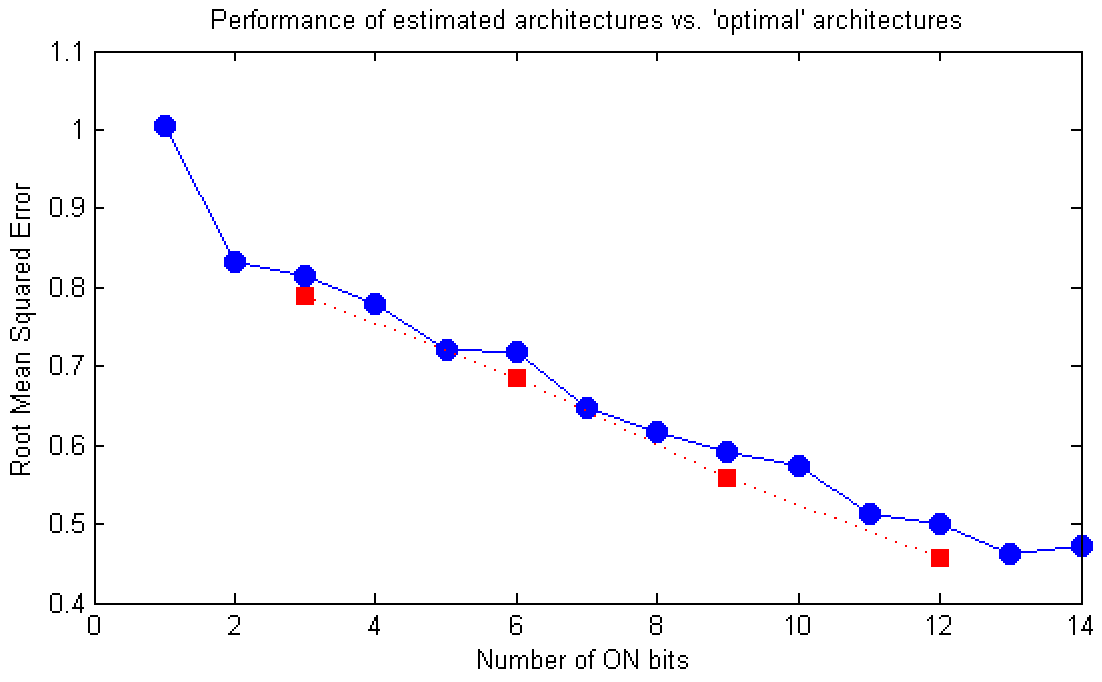

Figure 6.12: Performance differences between the 'optimal' architectures and estimated architectures

Figure 6.12 illustrates the performance of those 'optimal' NN models against those of the estimated architectures. The NN models of 'optimal' architectures are always of better performance over those with estimated architectures. However, the advantages of these 'optimal' architectures are minor in this application. A possible reason for these non-significant advantages in the ‘optimal’ architectures is the upper bound for the total number of hidden neurons, which is twice the number of input variables. All these ‘optimal’ architectures are either close to or equal to their upper bounds. Thus, lifting the upper bounds may lead to NN modes of even better performance.

6.7 Symbolic Regression using Genetic Programming

After constructing the parameter importance (influence) rank table, a GP with tree-structured MBE is employed to discover the symbolic relationship between various numbers of input parameters selected from the importance rank table and the logarithm of the corresponding overtopping discharges. As discussed in the previous sections 3.2.4 and 4.7, the GP uses seven commonly used mathematical operators, which are plus, minus, times, division, power, natural logarithm and square root, as the source of expressions to construct the symbolic expression of the relationship.

The GP symbolic regression exploration is carried out for various numbers (3, 6 and 9) of input parameters. For the case of 3 input parameters, the population size is estimated at 105 using the rule of thumbs expressed in equation 3.10, section 3.2.4.2, the maximum number of tree branches in a chromosome is 5, the initial maximum tree depth is 3, and the size of the elite group is estimated at 5 (5% of the total population). Figure 6.13 illustrates the evolution of the GP for 100 generations for the case of 3 input parameters selected from the rank table.

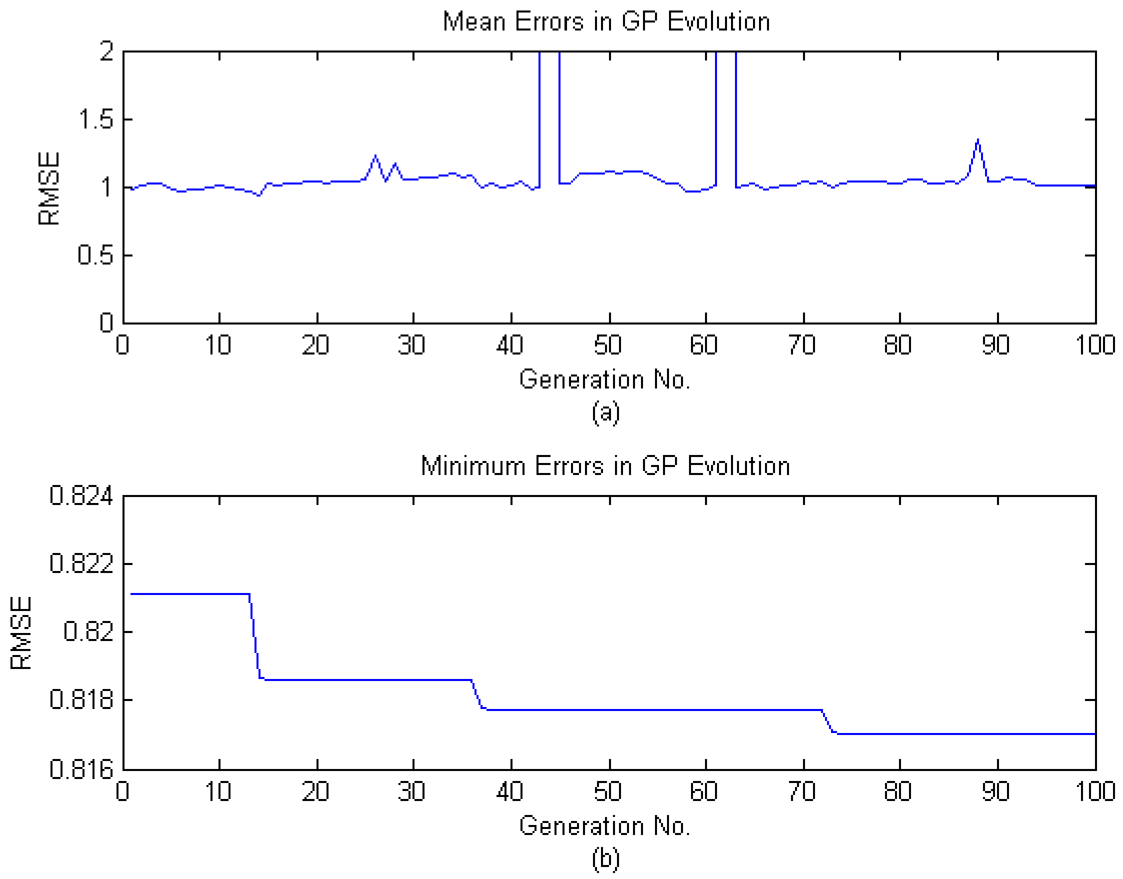

Figure 6.13: Evolution of GP for the case of 3 variables. (a) mean errors in generations, (b) minimum errors in generations

As shown in Figure 6.13(a), the mean generation RMSEs fluctuates around 1 with few sharp peaks and two extremely high values. Figure 6.13(b) illustrates the RMSEs of the best performing chromosome in each generation, 3 clear decreasing steps occur at the 14th, 37th and 73rd generations. However, the improvements made at these dropping points are minor. The global minimum RMSE occurred at the 73rd generation onwards is 0.8170 and the chromosome of minimum RMSE can be expressed using equation 6.7. Comparing with the NN models with the estimated and 'optimal' architectures that produce RMSEs at 0.8152 and 0.7881 respectively, this chromosome produces competitive performance.

$$y = A * x_2 + B * (x_2^2 - x_2) + C * x_3 + D * ((x_1)^{\frac{1}{5}})^{\frac{1}{4}} + E * x_1 * \log(x_3) + F \qquad (6.7)$$

Table 6.7 lists the performance of the GP process and the discovered symbolic expressions using 6 and 9 variables from the SGNO rank table. Comparing with the NN models using the same input variables, these chromosomes are not performing as well as the NN models. However, these chromosomes produce better performance than the chromosome using 3 variables.

Table 6.7: Symbolic expressions discovered by GP for various numbers of variables

No. of Variables	RMSE	Symbolic Expression
6	0.7725	$y = A * x_4 + B * \left(x_3^{-3}(x_4 - x_1)x_2 + (x_2x_3)^{\frac{1}{3}} + x_2\right)$ $+C * x_1 + D * (x_3 + x_5)(x_5 - x_4) + E$
9	0.7618	$y = A * \left((x_2 - x_6) - x_4{}^{\frac{1}{2}}\right)$ $+B * \left(LOG(x_5) - \frac{(x_1 - x_9 + x_4 + x_5)^2}{x_7}\right)^{-3}$ $+C * x_6 + D * \left(x_8 + \left(\frac{x_7 + x_7}{x_1}\right)^{\frac{1}{2}}\right) + E * x_2{}^{\frac{1}{2}} + F$

6.8 Benchmarking with other Feature Selection (FS) Techniques

In this section, the performance of SGNO is benchmarked with several other FS techniques, including PCA, FFS, BFS and GNMM. All these techniques are designed to reduce the dimensionality of the original data set.

6.8.1 Principal Component Analysis (PCA)

PCA is a famous dimension reduction technique that transforms the original data set in to a new coordinate system linearly. Section 3.3.1 explains the PCA process in details. In this application, PCA is used to transform the 14 input parameters and the performance of the new parameters, principal components (PCs), is evaluated using NNs. Figure 6.14 illustrates the variances carried in each PC after transformation. The figure displays only 4 PCs and the first PC carries the most variance, close to 80%, while the other PCs carry only small amounts of variances. The accumulated variance plot (the thin blue line) shows that the first 4 PCs displayed explain over 95% of the variance.

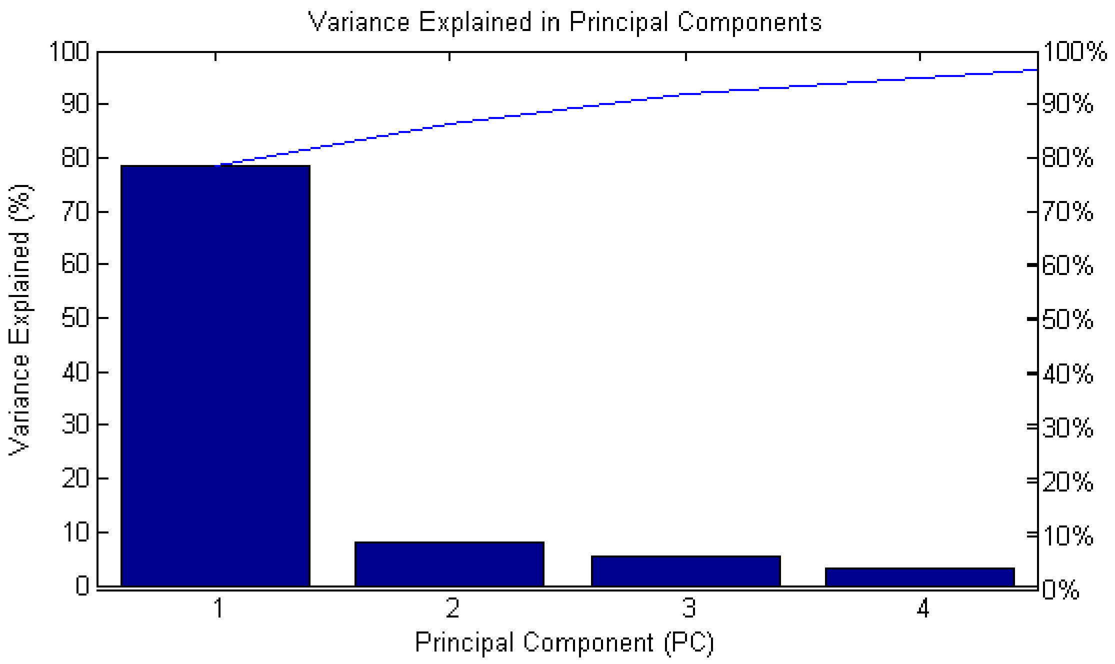

Figure 6.14: Variance explained in PCs

After the PCA transformation, the performance of the PCA is evaluated by employing NNs taking various numbers of PCs as inputs to estimate overtopping discharges. Again, five-fold cross-validation and early-stopping NN training is employed to support the NN evaluation. Furthermore, the NN models comprise of a single hidden processing layer and the number of hidden neurons is estimated by halving the number of inputs and output. Table 6.8 lists the performance of the NN models taking various number s of PCs as inputs and Figure 6.15 illustrates the performance of these NN models against the performance of NN models using input variables picked from the parameter rank table constructed using SGNO.

Table 6.8: PCA re-evaluation results

Size	1	2	3	4	5	6	7	8	9
RMSE	1.0997	1.1047	1.0887	1.0594	0.9919	0.9626	0.8699	0.6926	0.6253

Size	10	11	12	13	14
RMSE	0.6123	0.5848	0.5726	0.5391	0.475

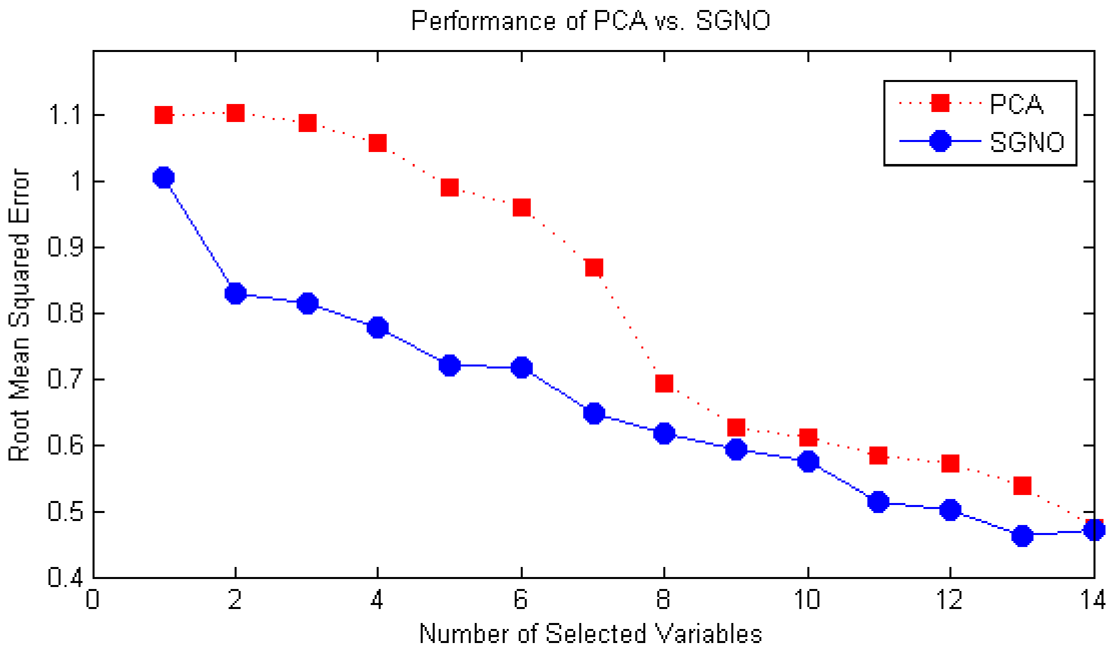

Figure 6.15: Performance of NN models using PCA against SGNO

As shown in Figure 6.15, the parameters selected by SGNO are of clear advantages over the PCA as the RMSEs generated using the SGNO selected parameters are always lower than those generated using PCA and the largest difference is up to the value of 0.29 when 4 input variables are used. Both performance plots have reducing patterns; as the number of selected variables increases, the difference between these two plots reduces indicating that the SGNO gradually loses its advantage.

6.8.2 Forward Feature Selection (FFS) and Backward Feature Selection (BFS)

FFS and BFS are well known conventional sequential FS techniques. They work towards the solution along two opposite directions. The FFS starts with an empty selection list and repetitively adds variables into the selection list, while the BFS starts with a selection list containing all the variables and repetitively removes the variables from the list. The detailed procedures of FFS and BFS are explained in section 3.3.2.

The following lists present the variables preference orders discovered by FFS and BFS. The FFS list presents variables in the sequential order that variables are added to the solution list, while the BFS list presents variables in the reversed order that they are removed from the selection in BFS. Hence, the variables are in descending order of preferences in both lists. In the lists, the first three parameters are identical and the other parameters are of various orders.

FFS: [10, 7, 4, 2, 9, 12, 14, 6, 5, 1, 11, 13, 3, 8]

BFS: [10, 7, 4, 9, 6, 12, 2, 14, 13, 8, 11, 5, 1, 3]

Comparing with the parameter importance rank table constructed by SGNO, which is **[7, 10, 1, 13, 14, 11, 8, 9, 6, 5, 4, 3, 12, 2]**, the parameters are ordered in distinct ways. However, they all identify the parameters No.7 and No.10 as the most preferable parameters and the parameter No.3 is less favourable. Table 6.9 lists the performance of the preference lists produced by BFS and FFS using NN models with various numbers of input variables selected from the preference lists.

Table 6.9: BFS and FFS re-evaluation results

Size	1	2	3	4	5	6	7	8	9
BFS	1.0041	0.8348	0.774	0.7414	0.6877	0.6432	0.5908	0.5747	0.5371
FFS	1.004	0.8343	0.7794	0.7428	0.6855	0.6465	0.5971	0.5812	0.5429

Size	10	11	12	13	14
BFS	0.5335	0.5011	0.5045	0.4795	0.4783
FFS	0.5363	0.5103	0.5072	0.4849	0.4788

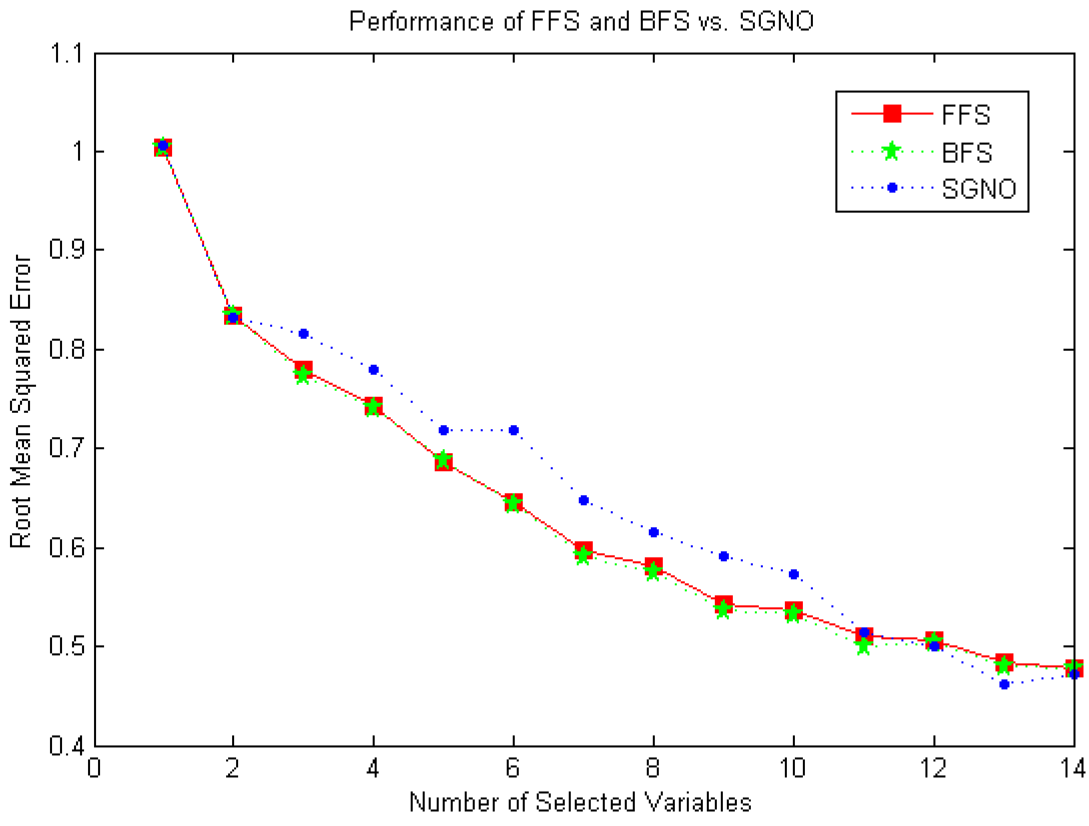

Figure 6.16: Performance of NN models using BFS and FFS against SGNO

Figure 6.16 illustrates the performance of the preference lists produced by BFS and FFS against rank table produced by SGNO. Although the BFS and FFS produced different parameter preference lists, the performance of the lists are almost the same as their performance plots are nearly overlapped. Comparing with the performance plots of BFS and FFS, the parameter rank table discovered by SGNO is not performing as well as BFS and FFS apart from the head and tail of the performance plot.

6.8.3 Genetic Neural Mathematical Method (GNMM)

GNMM is a hybrid data mining technique incorporating GA and NNs. The preferences of the available variables are ordered in terms of their appearances during the GA process. The variables of higher appearance percentages are considered to be more favourable compared to those of lower appearance percentages (Yang, 2010; Yang et al., 2007). The detailed procedures of GNMM were discussed in section 3.3.3. Figure 6.17 illustrates the appearance percentages of the variables discovered by the GNMM process.

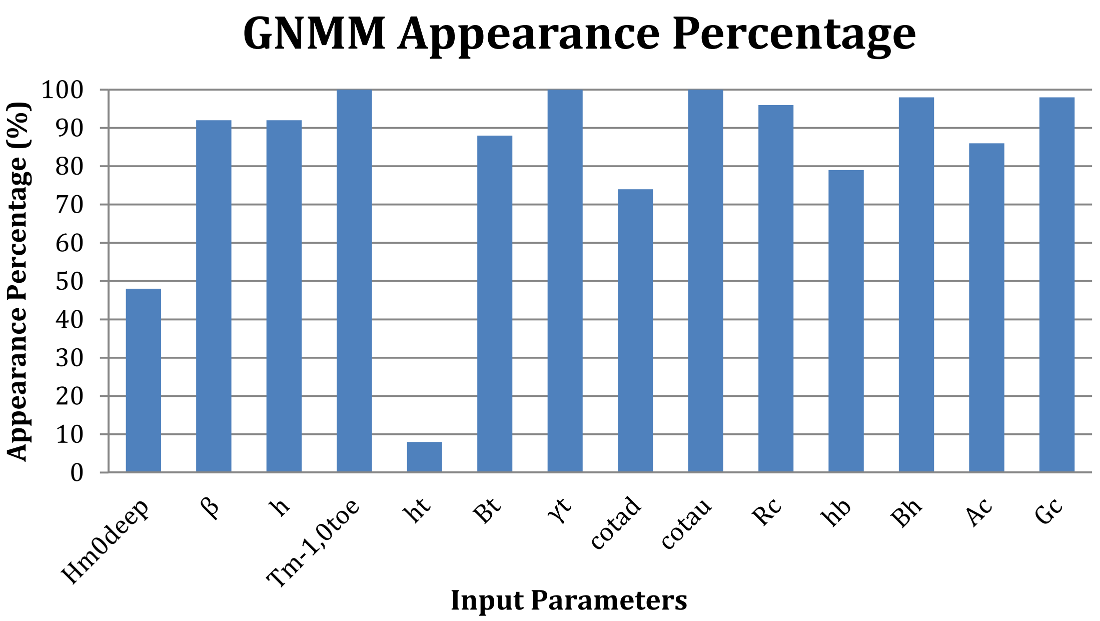

Figure 6.17: Variable appearance percentages in GNMM

As shown in Figure 6.17, most of the input parameters are of high appearance percentages (over 80%). Among all the parameters, $T_{m\text{-}1,0toe}$, γ_t and $cota_u$, indexed by 4, 7 and 9 respectively, are of the highest appearance percentages (close to 100%), followed by R_c (10), B_h (12), G_c (14), β (2) and h (3) (over 90%). The following list contains the ordered variables based on their appearance percentages in descending order.

[$T_{m\text{-}1,0toe}$, γ_t, $cota_u$, B_h, G_c, R_c, β, h, B_t, A_c, h_b, $cota_d$, H_{m0deep}, h_t]

The following index list holds the indices of the variables ordered in the above list.

[4, 7, 9, 12, 14, 10, 2, 3, 6, 13, 11, 8, 1, 5]

Table 6.10 lists the performance of the preference list produced by GNMM using NN models with various numbers of input variables selected from the list. Figure

6.18 illustrates the performance of the GNMM discovered parameter rank table against SGNO's rank table.

Table 6.10: GNMM re-evaluation results

Size	1	2	3	4	5	6	7	8	9
RMSE	1.1261	1.0437	0.9846	0.9763	0.9614	0.6764	0.6023	0.5826	0.5546

Size	10	11	12	13	14
RMSE	0.5448	0.5132	0.4965	0.4746	0.4798

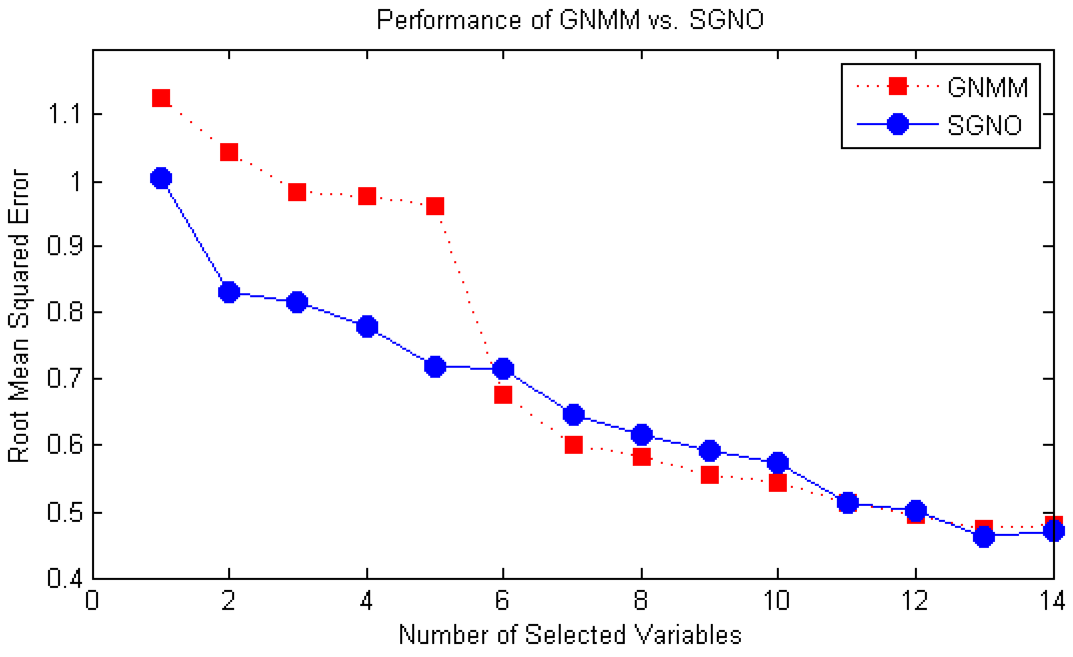

Figure 6.18: Performance of GNMM against SGNO

As shown in Figure 6.18, unlike the performance plot of SGNO, which decreases gradually following a quasi-linear pattern, there exists a clear gap in the middle of the plot (between 5 and 6). The sudden drop is caused by the addition of the 10th variable, which is the crest freeboard (R_c) and it is identified as the most preferable variable by BFS and FFS. The performance plot of GNMM shows that SGNO is of significant advantages over GNMM when the number of input parameters is less than or equal to 5. The largest difference is up to the value of

0.2418 when 5 input variables are used. When the number of parameters exceeds 5, the rank table constructed by GNMM generally performs better than SGNO.

6.8.4 Benchmarking summary

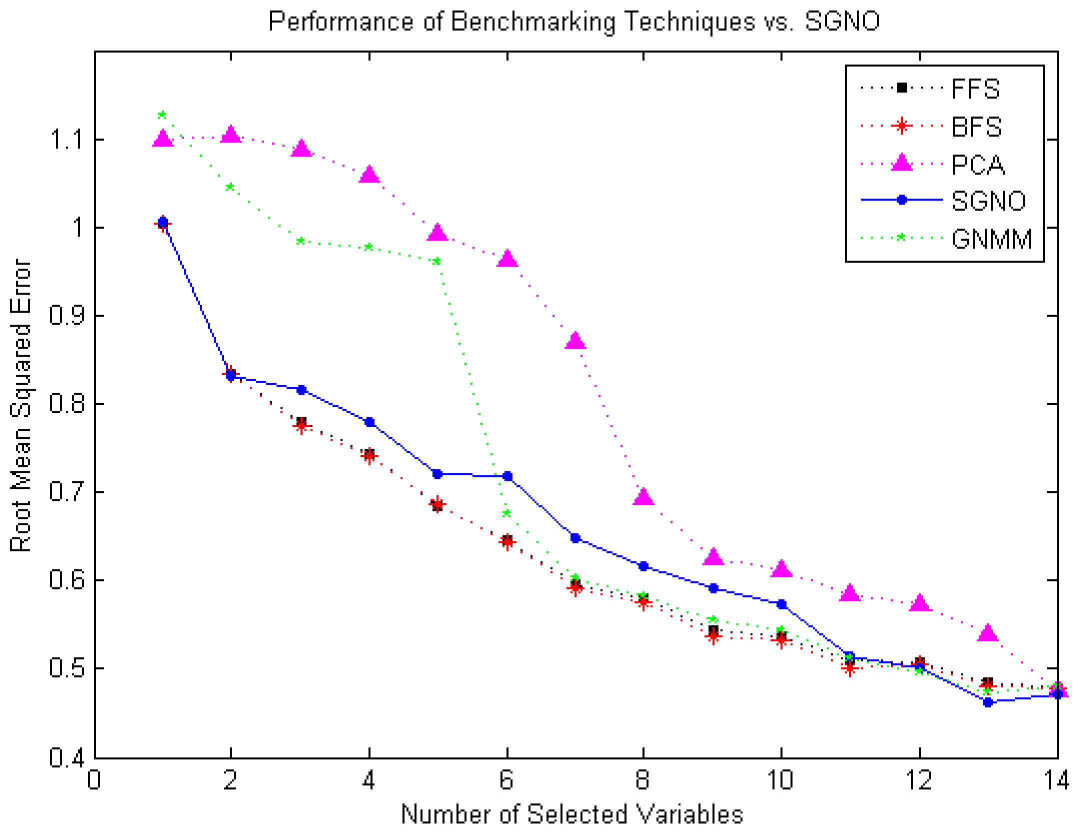

Figure 6.19: Performance of all benchmarking techniques against SGNO

Figure 6.19 summarises the performance of the preference rank tables constructed using all the techniques presented. It shows that among these 5 techniques, FFS and BFS are of the best performance and PCA generates the highest RMSEs in general. SGNO provides competitive performance to FFS and BFS, while GNMM is only performing well when the number of selected variables is relatively large (greater than 5).

6.9 Conclusion

In this chapter, SGNO is demonstrated to establish a model to predict the overtopping discharges at coastal structures using a database created in the European project CLASH. The database includes tests collected from different institutes and new tests performed within the CLASH project. Each test record contains a number of parameters representing wave characteristics and structural information at the test station plus some general information. Apart from establishing prediction models, the symbolic relationship between the model inputs and output are discovered using multiple-branch GP.

Analysis results show that all variables contribute to the prediction of overtopping discharge as increasing the number of input variables can always increase the accuracy of the NN prediction model (see Figures 6.6 and 6.18). SGNO successfully constructs the influence/importance rank table for all the variables. Comparing with 4 benchmarking FS techniques, including BFS, FFS, GNMM and PCA, SGNO is of clear advantage over GNMM and PCA, and of similar performance to the best performing techniques, BFS and FFS, especially when the number of selected variables is small.

After identifying the influential levels of all the variables, GP with tree structured multiple branches is applied to extract concise symbolic relationship between various numbers of input variables and the overtopping discharges.

References

Ahrens J.P., Heimbaugh M.S., Davidson D.D. (1986) Irregular wave overtoping of seawall/revetment configurations, roughans point, massachusetts, Department of The Army, Vicksburg, Mississippi.

Cybenko G. (1989) Approximations by superpositions of sigmoidal functions. Mahtematics of Control, Signals and Systems 2:303-314.

De Rouck J., Verhaeghe H., Geeraerts J. (2008) Crest level assessment of coastal structures - General overview. Coastal Engineering 56:99-107.

Dunteman G.H. (1989) Principal Components Analysis SAGE Publications.

Goda Y. (2009) Derivation of unified wave overtopping formulas for seawalls with smooth, impermeable surfaces based on selected CLASH datasets. Coastal Engineering 56:385-399.

Ingram D.M., Gao F., Causon D.M., Mingham C.G., Troch P. (2009) Numerical investigations of wave overtopping at coastal structures. Coastal Engineering 56:190-202.

Mase H., Sakamoto M., Sakai T. (1995) Neural network for stability analysis of rubble mound breakwaters. Journal of Waterway, Port, Coastal, and Ocean Engineering 121:294-299.

Medina J.R. (1999) Neural network modelling of runup and overtopping, International Conference on Coastal Structures, Santander, Spain. pp. 421-429.

Medina J.R., Gonzalez-Escriva J.A., Garrido J., De Rouck J. (2002) Overtopping analysis using neural networks, The 28th International Conference of Coastal Engineering, ASCE, Cardiff, United Kingdom. pp. 2165-2177.

Owen M.W. (1980) Design of seawalls allowing for wave overtopping, HR Wallingford.

Pozueta B., van Gent M.R.A., van den Boogaard H.F.P. (2004) Neural Network prediction, CLASH WP8 Neural Network, Delft Hydraulics, The Netherlands.

Saville T.J. (1955) Laboratory data on wave run-up and over-topping on shore structures, Beach Erosion Board, US Army Corps of Engineers, Vicksburg. pp. 32.

Soliman A.S.M. (2003) Numerical Study of Irregular Wave Overtopping and Overflow, School of Civil Engineering, University of Nottingham, Nottingham. pp. 242.

van Gent M.R.A., Pozueta B., Van den Boogaard H.F.P., Medina J.R. (2005) Final report on generic prediction method, Delft hydraulics. pp. 33.

van Gent R.A., van den Boogaard H.F.P., Pozueta B., Medina J.R. (2007) Neural network modelling of wave overtopping at coastal structures. Coastal Engineering 54:586-593.

Verhaeghe H. (2005) Neural Network Prediction of Wave Overtopping at Coastal Structures, Universiteit Gent, Gent. pp. 358.

Verhaeghe H., De Rouck J., van der Meer J. (2008) Combined classifier-quantifier model: A 2-phases neural model for prediction of wave overtopping at coastal structures. Coastal Engineering 55:357-374.

Yang J.H. (2010) Intelligent data mining using artificial neural networks and genetic algorithms: techniques and applications, Engineering, University of Warwick, Coventry. pp. 260.

Yang J.H., Hines E.L., Iliescu D.D., Leeson M.S. (2007) GNMM and accurate longitudinal dispersion coefficient prediction, The Seventh UK Chinese Association of Resource and Environment (CARE) Annual Meeting, Greenwich, London.

CHAPTER 7

Industry Level Production Growth Prediction and Modelling using the EU KLEMS Database

7.1 Overview

In the previous chapter, SGNO was applied to analyse the dataset collected in the EU CLASH project. In this chapter, SGNO is used to the EU KLEMS database to model the relationship between growths of the industrial inputs, such as capital, labour, energy, etc., and the growth of the gross production output.

The continuous improvement of economic growth and industrial production has been a major research area for decades. Empirical and theoretical research in the related fields, such as the relationship between technological improvement, employee's skill, investment and productivity have been carried out by researchers and economists. Typically, economic growth is primarily driven by the industrial productivity, which refers to the production of goods or services with the same amount of labour inputs, capital inputs, energy and material resources. To overcome the lack of a readily available database for internationally comparable studies on economic growth, a European project named EU KLEMS (capital, **K**, labour, **L**, energy, **E**, materials, **M**, and service, **S**) was established by the European Commission to create a general purpose database on growth and productivity accounts by industry for EU member states.

7.2 Background

Economists have been applying NNs to solve economic problems for decades by adopting the NNs for economic classification and regression problems (Herbrich et al., 1999). In the 1990's, many successful attempts using NNs on economic classification problems were published in the area of bankruptcy prediction. Odom and Sharda (1990) built a simple back propagation NN of a single hidden processing layer, which takes five financial ratios as the inputs. They reported that the NN model can achieve 70%-80% accuracy on the test dataset. Tam and Kiang (1992) employed a similar NN, which takes 19 financial ratios as inputs, and showed that the NN model performed on average better than other traditional financial methods. In the following years, many economists carried out similar tests using various types of NNs on different datasets. Boritz and Kennedy (1995) concluded in their study that the NNs performed reasonably well in bankruptcy prediction but their performance is affected by many factors, such as choice of predictor (input) variables, sample size, mixture proportion, etc (Zhang et al., 1999).

NN regression is of great interest to economists to perform economic forecasting. Macroeconomic forecasting is difficult due to the lack of an accurate and convincing model and the non-stationarities, high noise levels and nonlinear effects in the economic data series (Moody, 1995). As the NNs are capable of approximating any continuous functions, they are expected to provide effective nonlinear models and thus better predictions (Zhang et al., 1999). Moody (1995) presented his attempt to forecast the U.S. Industrial Production Index using NNs and demonstrated that NN models can achieve superior accuracy over conventional linear time series and regression methods. In 1996, Kaastra and Boyd (1996) introduced an 8-step practical introductory guide for economic researchers to design NNs for forecasting economic time series data. Junoh (2004) presented a comparative case study between NNs and linear econometric methods to predict the GDP growth in Malaysia using the economy indicators, such as number of internet subscribers, number of mobile phone subscribers, etc., collected from 1995-2000 and demonstrated that the NN model is of clear advantage over the econometric methods. Duzgun (2010) carried out comparative studies on Consumer Price Index inflation forecasting using generalized regression NNs (GRNNs), feed forward NNs (FFNNs) and ARIMA (AutoRegressive Integrated Moving Average) models. The monthly data for the time period 2000-2008 provided by the Turkish Statistical Institute were used. The advantage of using GRNN over traditional FFNN and ARIMA was demonstrated in his study.

7.3 Dataset

As mentioned in the previous section, the EU KLEMS project was established to improve the international economic studies of the relationship between labour skill formation, investment, technology development and manufacturing production by creating a standard economic database at the industry level covering a large set of countries (O'Mahony and Timmer, 2009). The main objectives of the EU KLEMS project were (Koszerek et al., 2007):

- To create a sustainable growth, employment and productivity database at the industry level for the EU Member States and some non-EU countries.

- To make a methodological breakthroughs in the field of industry productivity measurement.
- To work towards an internationally comparable comprehensive, industry level statistics.

The established EU KLEMS database covers measures of industry output, input, growth, and derived variables, such as price indexed measures, developed for 30 countries (25 individual EU member states, the US, Japan, Korea, Canada and Australia) from 1970 onwards. The input measures for every country include various categories of capital (K), labour (L), energy (E), material (M) and service inputs (S). In the database, industries are divided into several industry sectors, which are groups of industries defined by the EU Commission. The database not only provides industry measures at the detailed industry level but also provides higher level aggregates for the industry sectors.

In this application, SGNO and NNs are applied to explore the relationship between the gross industry output (GO) growth and the economic input growth, including labour, capital and intermediate inputs using the industry measures from 15 well studied countries, in which the industry measures are fully available for the period 1970 – 2007. These counties include Australia, Austria, Czech Republic, Denmark, Finland, France, Germany, Italy, Japan, Netherlands, Slovenia, Spain, Sweden, UK and US.

Instead of studying the industry level measures, the aggregate industry sector based measures are taken into account due to the fact that the industry level measures are incomplete for some countries. Tables 7.1 and 7.2 list the industry sectors and the industry measures from these industry sectors to be studied.

Table 7.1: Major industry sectors

Index	Industry Sectors
1	Agriculture, hunting, forestry and fishing
2	Mining and quarrying
3	Manufacturing
4	Electricity, gas and water supply
5	Construction
6	Wholesale and retail trade
7	Hotels and restaurants
8	Transport and storage and communication
9	Finance, insurance, real estate and business services
10	Community social and personal services

Table 7.2: Industry measures

Industry measure	Abbreviation	Unit
Intermediate inputs	II	Millions local currency
Number of persons engaged	EMP	Thousands
Total hours worked by persons engaged	H_EMP	Millions of hours
Labour compensation	LAB	Millions local currency
Capital compensation	CAP	Millions local currency

As the industry measures were taken on all the industry sectors, Tables 7.1 and 7.2 produce a total of 50 variables, known as economic inputs or industrial inputs. Hence, the objective of this application is to model the relationship between the annual growth of GO and the growth of these 50 variables using NNs. For each of the countries selected, the 50 economic input variables and the GO output in each year are extracted to be analysed. As mentioned previously, the industry measures were collected for the period 1970 – 2007 (38 years). For the 15 selected countries, there should be 570 (15x38) data records and each record contains 51 variables representing the economic inputs and output for a country in a specific year. However, the industry measures for several countries may not be complete, i.e. the measures of early decades are not available. Hence, the extracted data subset contains only 508 records.

7.4 Data Pre-processing

As this application aims to model the relationship between the growths on economic inputs and the growth on gross production output using NNs and analyse the effects of the input growth on the growth of the output, the original industry measures (economic inputs and production output) extracted need to be converted into the annual growth rates using the follow equation:

$$\text{growth rate} = \text{meansure}_{n+1} / \text{measure}_n, \quad n = 1970, 1971, \ldots \tag{7.1}$$

Converting the industry measures to industry growths reduces the size of the extracted dataset down to 493. Table 7.3 list the basic statistics of the selected 51 variables. The variable GO is the gross production growth, the output in NN models, and the other variables (II_n, EMP_n, H_EMP_n, LAB_n and CAP_n) are the inputs in NN models, representing the various types of economic input growths in those selected industry sectors (the subscripts indicates the industry sectors listed in Table 7.1). The GA index in the table represents the position of the variable in GA chromosomes during the SGNO process.

Table 7.3: Statistics of the selected variables

GA Index	Symbol	Mean	Minimum	Maximum	Standard Deviation
	GO	1.0803	0.9491	1.3506	0.0528
1	II_1	1.0474	0.6462	1.5590	0.0798
2	II_2	1.0807	0.6415	1.7912	0.1460
3	II_3	1.0736	0.8877	1.5078	0.0753
4	II_4	1.0965	0.4465	1.7983	0.1343
5	II_5	1.0773	0.8034	1.3864	0.0796
6	II_6	1.0853	0.9167	1.5357	0.0718
7	II_7	1.0919	0.8197	2.4536	0.1273
8	II_8	1.0966	0.9249	1.5658	0.0776
9	II_9	1.1052	0.8008	1.6069	0.0740
10	II_{10}	1.0934	0.8633	1.5300	0.0693
11	EMP_1	0.9750	0.7730	1.0963	0.0289
12	EMP_2	0.9758	0.5738	1.2220	0.0588
13	EMP_3	0.9910	0.9000	1.0589	0.0251
14	EMP_4	0.9971	0.8257	1.1481	0.0353
15	EMP_5	1.0033	0.8462	1.1248	0.0427
16	EMP_6	1.0091	0.8866	1.0768	0.0201
17	EMP_7	1.0196	0.9006	1.0996	0.0268
18	EMP_8	1.0068	0.9359	1.0717	0.0192
19	EMP_9	1.0364	0.9262	1.1654	0.0298
20	EMP_{10}	1.0196	0.9544	1.0744	0.0149
21	H_EMP_1	0.9747	0.8385	1.1331	0.0327
22	H_EMP_2	0.9745	0.4577	1.3030	0.0687
23	H_EMP_3	0.9888	0.8792	1.0648	0.0283
24	H_EMP_4	0.9945	0.8216	1.1481	0.0367
25	H_EMP_5	1.0020	0.8296	1.1405	0.0471
26	H_EMP_6	1.0048	0.8943	1.0827	0.0227
27	H_EMP_7	1.0139	0.8748	1.1089	0.0295
28	H_EMP_8	1.0050	0.9238	1.0735	0.0221
29	H_EMP_9	1.0334	0.9398	1.3059	0.0330
30	H_EMP_{10}	1.0168	0.9486	1.0927	0.0172
31	LAB_1	1.0439	0.8484	1.4491	0.0796
32	LAB_2	1.0500	0.3955	2.2275	0.1232
33	LAB_3	1.0621	0.9328	1.3977	0.0586
34	LAB_4	1.0688	0.8855	1.4793	0.0763
35	LAB_5	1.0731	0.8385	1.3180	0.0708
36	LAB_6	1.0758	0.9101	1.3184	0.0608
37	LAB_7	1.0811	0.9049	1.4879	0.0679
38	LAB_8	1.0732	0.9486	1.3343	0.0585
39	LAB_9	1.1047	0.9418	1.3544	0.0636
40	LAB_{10}	1.0869	0.9540	1.3940	0.0614
41	CAP_1	0.8405	-126.3476	27.2510	6.0233

42	CAP_2	1.1006	-6.6127	5.8661	0.5701
43	CAP_3	1.0917	0.4237	5.8732	0.2721
44	CAP_4	1.1810	0.0429	46.7234	2.0598
45	CAP_5	2.2875	-165.7900	735.6785	33.9707
46	CAP_6	1.1023	-5.1708	3.6209	0.4092
47	CAP_7	1.0332	-15.4249	16.8087	1.6928
48	CAP_8	1.1040	-2.4528	14.7725	0.6693
49	CAP_9	1.0975	0.6827	1.9848	0.0817
50	CAP_{10}	1.0994	0.7755	2.0325	0.1075

As the SGNO employs five-fold cross-validation, the derived dataset of 493 records is divided randomly into five groups. Table 7.4 shows the distribution of these five groups.

Table 7.4: Sample distributions in five-fold cross-validation

	Set A	**Set B**	**Set C**	**Set D**	**Set E**	**Total**
Count	99	98	99	98	99	493

7.5 Variable Selection using Sensitivity Genetic Neural Optimisation (SGNO)

7.5.1 Neural Network module

In this application, the NN module is used to establish the relationship between the proposed input variable group, represented as chromosomes in the GA module, and the annual gross production growth. Similar to the application described in the previous chapters, the NN module employs five-fold cross-validation with early-stopping. The NNs are of single hidden processing layer and the number of hidden neurons in the processing layer is estimated by halving the sum of input and output variables. The initial weights in the hidden neurons are random numbers. The activation functions in the hidden layer are the commonly used tangent-sigmoid functions (see Equation 2.5) and the activation function in the output layer/neuron is the pure linear function (see Equation 2.2).

7.5.2 Genetic Algorithm module

In this application, the size of each binary chromosome in the GA module is 50 as there are 50 input variables extracted from the EU KLEMS database. The size of the GA population is estimated at 250 using the rule of thumb discussed previously (see Section 3.2.2.2 and Equation 3.3) and the size of the elite group is estimated at 12, which is about 5% of the total population. Figure 7.1 illustrates the performance of the chromosomes evaluated in 50 GA generations. It clearly shows that the mean RMSE in each generation drops rapidly in the first few generations followed by a steady trend of convergence. Figure 7.2 illustrates the size of the chromosomes (number of ON bits) in the GA generations.

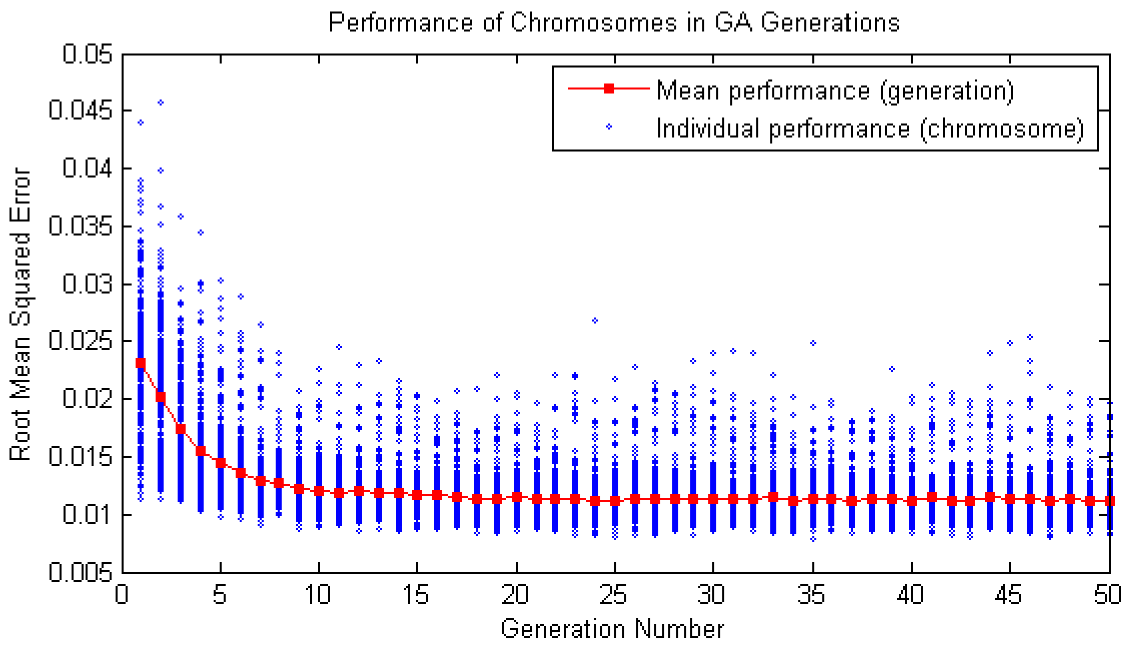

Figure 7.1: Chromosomes performance in GA generations

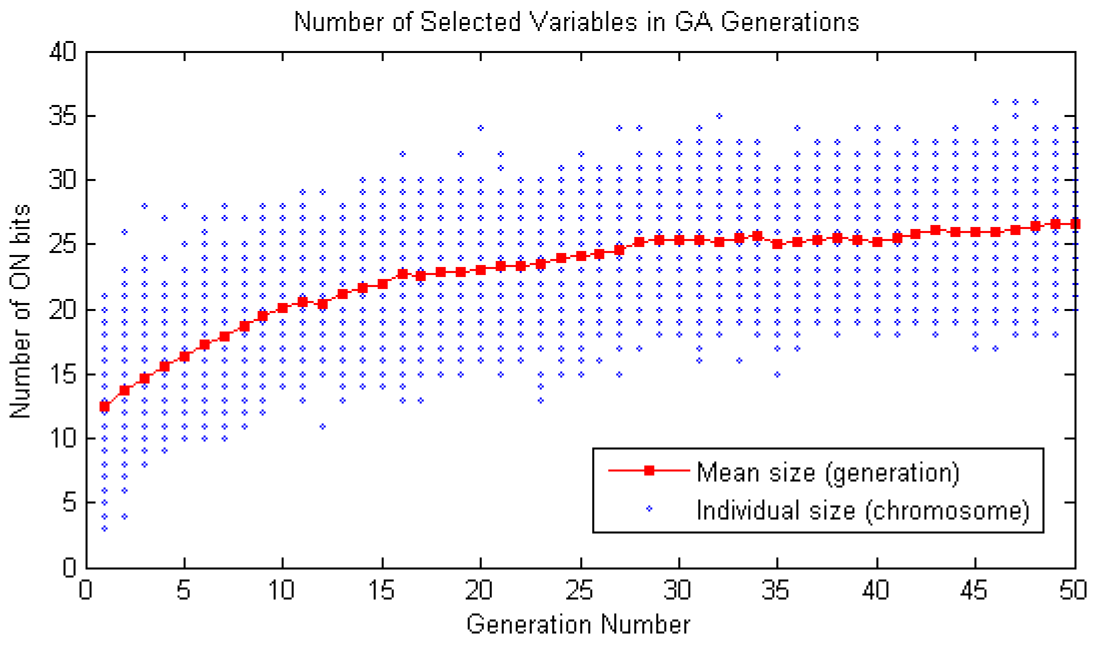

Figure 7.2: Number of ON bits in each generation

It is clear in Figure 7.2 that the average size of the chromosome in the GA generations gradually increases. Unlike the pattern showed in Figure 7.1, which is a rapid reduction followed by a steady flat line, the chromosome size keeps increasing but the rate of increase slows down as the GA progresses. Figure 7.3 illustrates the performance of chromosomes evaluated in GA against their sizes. It is can be seen that the RMSEs of the chromosomes decrease as the number of ON bits (size of chromosomes) increases until the size of chromosomes exceeds 20, beyond which the RMSE plot shows a clear converged pattern.

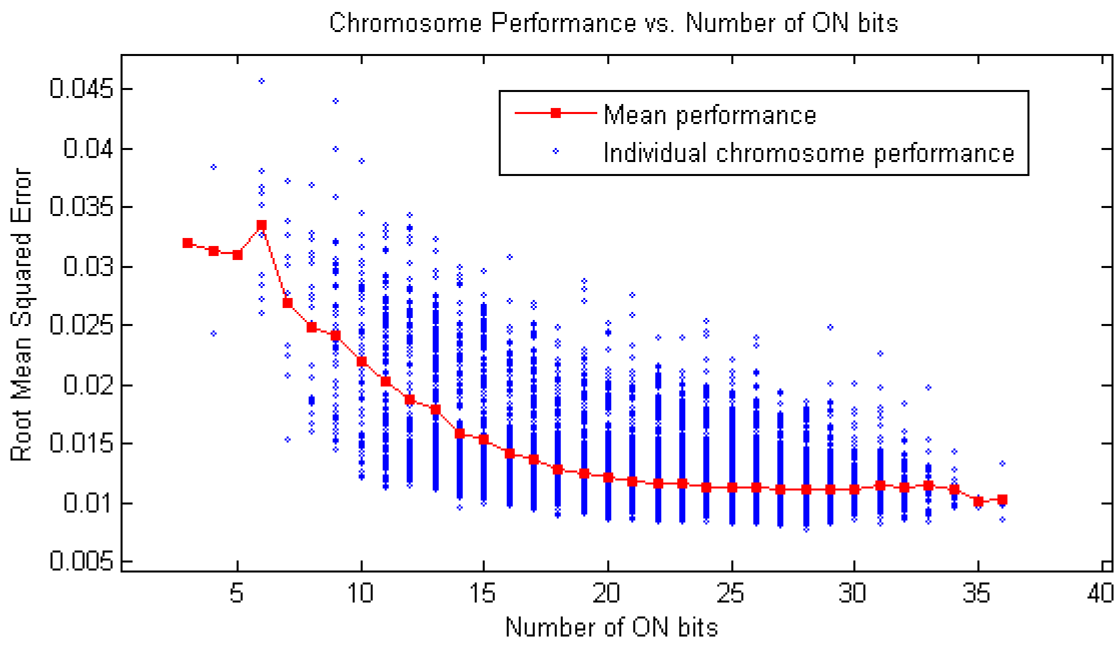

Figure 7.3: Chromosome performance vs. number of ON bits

7.5.3 Sensitivity Analysis module

After evaluating the proposed chromosomes in the GA module, the SA module is invoked to identify the global influences of the input parameters towards the output parameter by analysing the importance of the input parameters appeared in the group of selected well-performing chromosomes in each generation.

In this application, the GA module evaluated 250 chromosomes for 50 generations, a quarter of the chromosomes in the population (250 x 25% ≈ 62) are selected in each generation to carry out the SA analysis. Figure 7.4 illustrates the distribution of the sizes (number of ON bits) of the selected chromosomes. The figure shows that most of the chromosomes selected are medium-sized containing about 25 ON bits (50% of the total number of input variables).

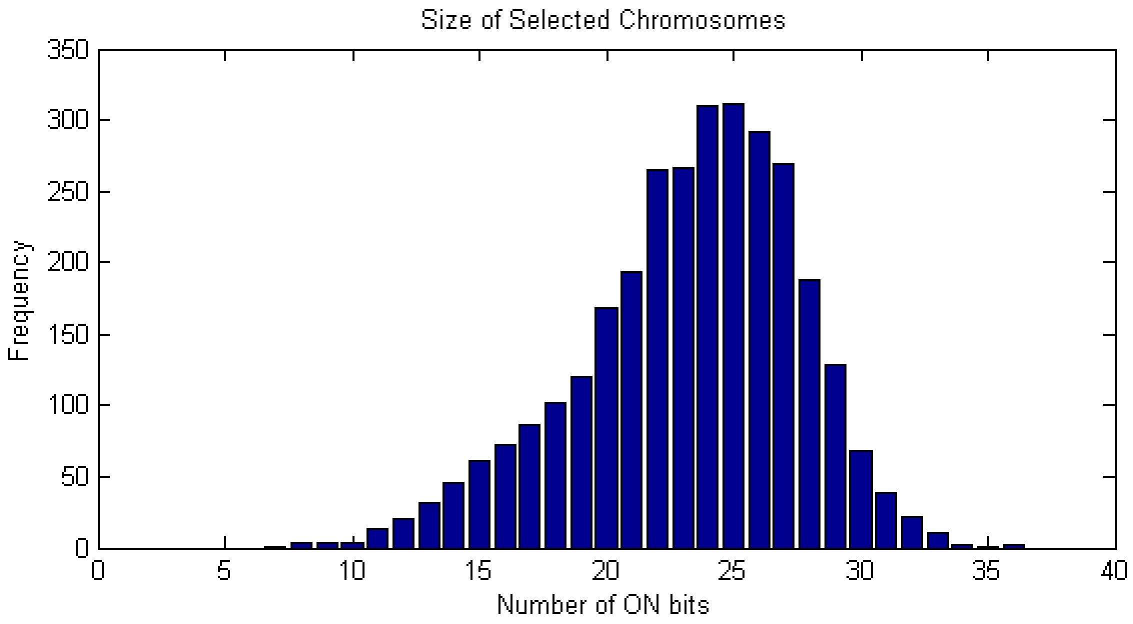

Figure 7.4: Sizes of the chromosomes selected for SA analysis

The global sensitivity measure of a certain input variable is calculated by taking the mean of the sensitivity measures derived in all selected chromosomes. Figure 7.5 illustrates the global sensitivity measures of all the input parameters. As shown in the figure, the intermediate input of the manufacturing sector, $\mathbf{II_3}$, the number of persons working at the community social services sector, $\mathbf{EMP_{10}}$, and the total working hours of the community social services sector, $\mathbf{H\_EMP_{10}}$, have significantly higher sensitivity measures comparing to other parameters. Several capital compensations, $\mathbf{CAP_n}$, have the lowest sensitivity measures and hence have the lowest influences on the model output.

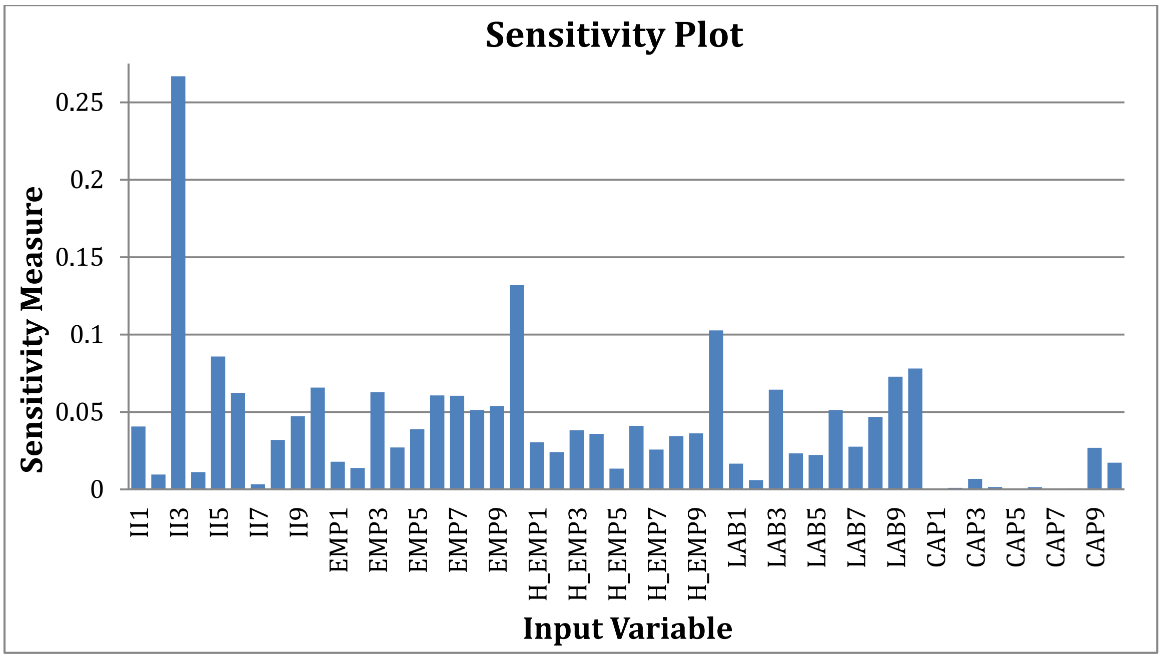

Figure 7.5: Global sensitivity measures of all input parameters

Based on the global sensitivity measures of the parameters, the importance rank table can be constructed by rearranging the parameters in descending order of their sensitivity measures. The variable of higher sensitivity measure is considered to be of higher importance/influence to the output. The importance rank table is listed below, in which the former elements are of higher sensitivity measures than the latter elements and are thus more important.

[II_3, EMP_{10}, H_EMP_{10}, II_5, LAB_{10}, LAB_9, II_{10}, LAB_3, EMP_3, II_6, EMP_6, EMP_7, EMP_9, EMP_8, LAB_6, II_9, LAB_8, H_EMP_6, II_1, EMP_5, H_EMP_3, H_EMP_9, H_EMP_4, H_EMP_8, II_8, H_EMP_1, LAB_7, EMP_4, CAP_9, H_EMP_7, H_EMP_2, LAB_4, LAB_5, EMP_1, CAP_{10}, LAB_1, EMP_2, H_EMP_5, II_4, II_2, CAP_3, LAB_2, II_7, CAP_4, CAP_6, CAP_2, CAP_8, CAP_7, CAP_1, CAP_5]

In terms of parameter indices in the GA chromosomes, the importance rank table can be expressed as

[3, 20, 30, 5, 40, 39, 10, 33, 13, 6, 16, 17, 19, 18, 36, 9, 38, 26, 1, 15, 23, 29, 24, 28, 8, 21, 37, 14, 49, 27, 22, 34, 35, 11, 50, 31, 12, 25, 4, 2, 43, 32, 7, 44, 46, 42, 48, 47, 41, 45]

7.6 System Remodelling

As mentioned in section 3.2.3, NNs are not only used as the NN module in SGNO, but also used to re-evaluate the performance of the ranked variables in the importance rank table. Table 7.5 lists the performance of NN models taking various numbers of variables from the importance rank table as the input variables (the selection of parameters always starts from the first element in the rank table) and figure 7.6 illustrates the results of NN models of 3, 6 and 9 input variables. Figure 7.7 illustrates the performance of these NN models against the chromosomes evaluated in the GA module.

Table 7.5: SGNO re-evaluation results

Size	1	2	3	4	5	6	7	8	9
RMSE	0.0251	0.0238	0.0252	0.0198	0.0145	0.0126	0.013	0.0129	0.0126
R^2	0.7744	0.7968	0.7723	0.8595	0.9244	0.9426	0.9397	0.94	0.9432

Size	10	11	12	13	14	15	16	17	18
RMSE	0.0114	0.011	0.0113	0.0113	0.0119	0.0114	0.0108	0.0115	0.0109
R^2	0.9535	0.9564	0.9542	0.9542	0.9492	0.9533	0.9579	0.9523	0.9573

Size	19	20	21	22	23	24	25	26	27
RMSE	0.0106	0.0105	0.01	0.0104	0.0105	0.01	0.0097	0.0107	0.0109
R^2	0.9595	0.9603	0.964	0.9614	0.9608	0.9639	0.9661	0.959	0.9575

Size	28	29	30	31	32	33	34	35	36
RMSE	0.0103	0.0097	0.0105	0.0107	0.0099	0.0098	0.0104	0.0106	0.0103
R^2	0.9623	0.966	0.9606	0.9588	0.9649	0.9654	0.9609	0.9598	0.9622

Size	37	38	39	40	41	42	43	44	45
RMSE	0.0099	0.0107	0.0106	0.01	0.01	0.0093	0.01	0.0111	0.0113
R^2	0.9648	0.9589	0.9597	0.9641	0.9638	0.9689	0.9642	0.9561	0.9542

Size	46	47	48	49	50
RMSE	0.011	0.0114	0.0117	0.0128	0.0169
R²	0.9569	0.9532	0.951	0.9411	0.8972

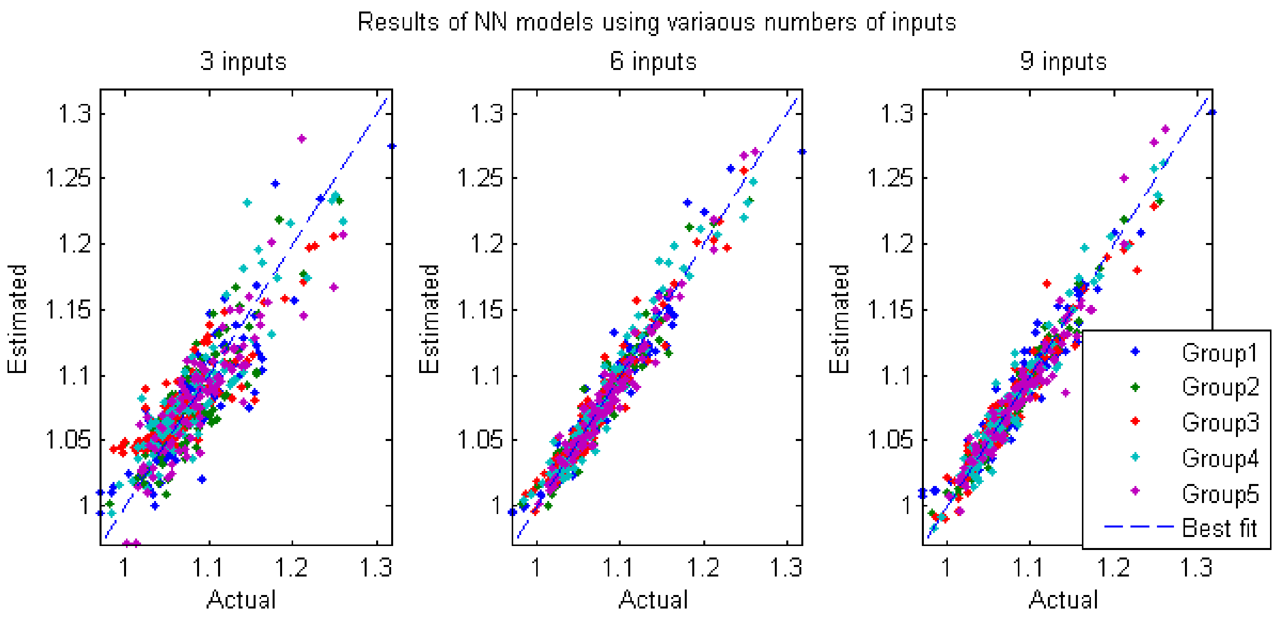

Figure 7.6: Performance of NN models using 3, 6 and 9 input parameters

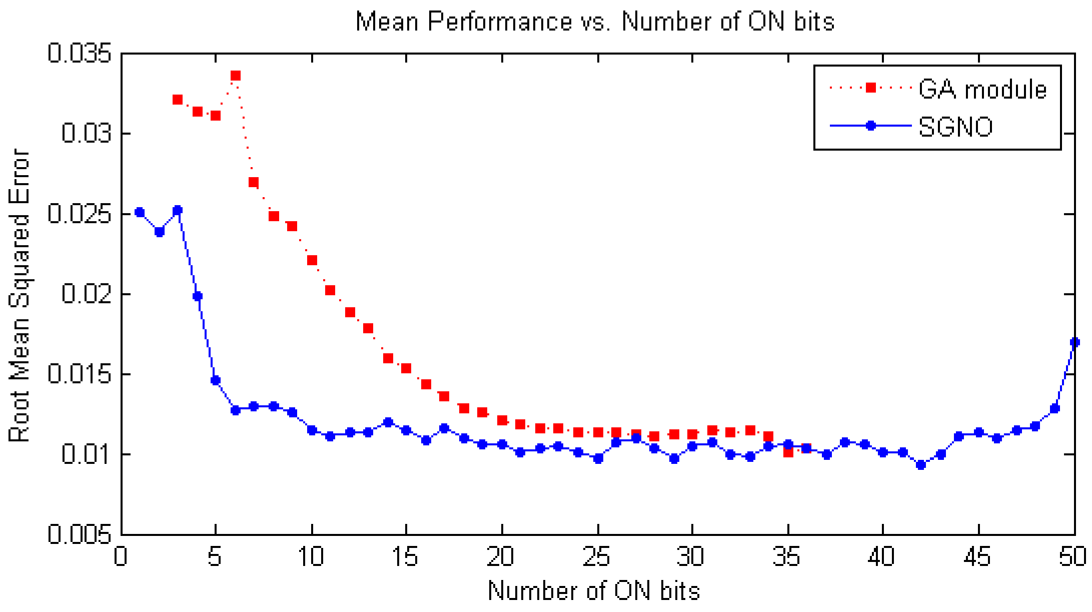

Figure 7.7: Performance of parameters in SGNO rank table against parameters in GA module

Figure 7.7 clearly shows the decreasing patterns of both plots, and the performance of the parameters selected by SGNO can be seen to be of significant advantage over the mean performance of the chromosomes evaluated during the GA process. The SGNO plot starts with a rapid reduction followed by a converging trend fluctuating around 0.011. As the number of variables (number of ON bits) increases, the advantage of SGNO optimisation weakens. At the end of the SGNO plot, where the number of ON bits is greater than 40, the plot rises again. These suggest that the first 6 to 10 variables in the importance rank table significantly contribute to the generation of the output variable in the NN models, and these variables are sufficient to estimate the output variable. The variables in the middle portion of the importance rank table are of minor, or even negligible, contribution in the NN models. Furthermore, the variables at the tail of the rank table are of negative contributions, i.e. these variables decrease the performance of the NN models.

To discover the best performance, i.e. the lowest RMSE, achievable using the variable ranking table, the popular trial and error approach is employed to find the NN models of the 'optimal' architectures at some key points in the importance rank table (3 parameters, 6 parameters, 9 parameters and 12 parameters) (Ahmed, 2005). Table 7.6 lists the 'optimal' architectures discovered for various numbers of parameters picked from the importance rank table. Figure 7.8 illustrates the performance of those 'optimal' NN models versus the estimated architectures. It can be seen that the NN models of 'optimal' architectures are always of better performance. However, these 'optimal' architectures only produce minor advantages, usually around 0.001, compared with those of estimated architectures.

Table 7.6: 'Optimal' architectures and their performance

Number of input variables	‘Optimal’ architecture [1st layer, 2nd layer]	Performance (RMSE)
3	[2, 0]	0.0241
6	[5, 0]	0.0116
9	[6, 0]	0.0116
12	[18, 4]	0.0106

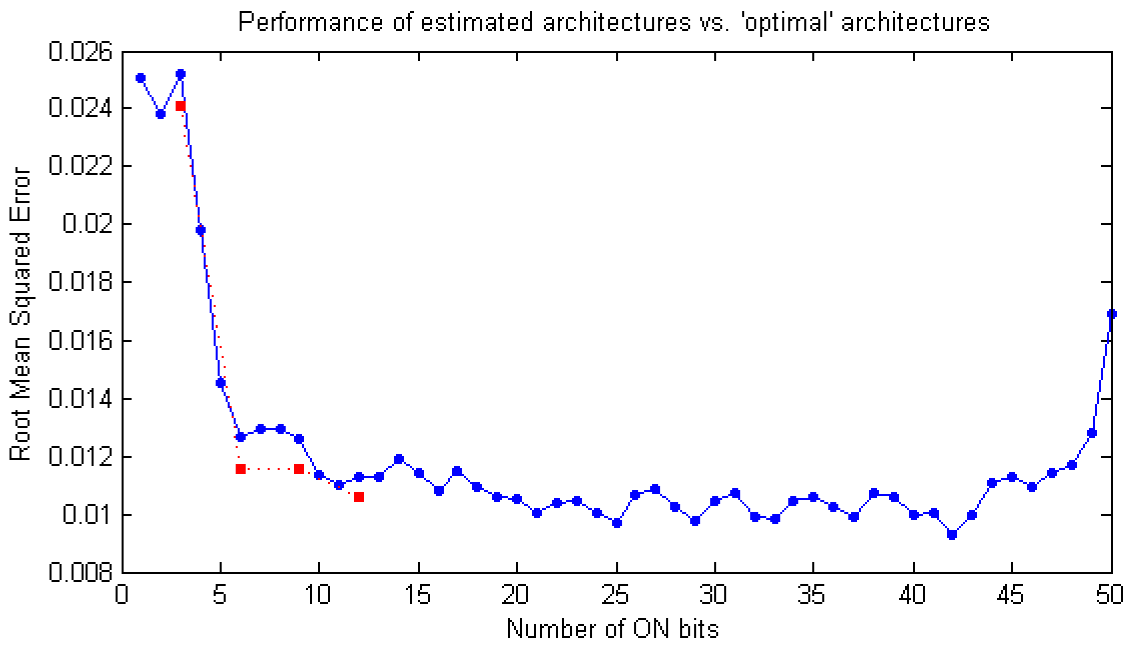

Figure 7.8: Performance differences between the ‘optimal’ architectures and estimated architectures

7.7 Symbolic Regression using Genetic Programming

In this section, tree-structured MBE GPs are used to find the symbolic relationship between input variables and the corresponding output. As discussed in the previous sections (see Sections 3.2.4 and 4.7), the MBE GP uses seven commonly used mathematical operators to construct the symbolic relationship between the selected input variables and the output variable. GP symbolic regression is carried out at several key points in this application, where various numbers (3, 6 and 9 out of 50) of variables are selected from the variable rank table constructed in SGNO (see section 7.5.3). For the case of 3 variables, the population size is estimated at 105 using the rule of thumbs (Equation 3.10) discussed in section 3.2.4.2; the maximum number of expression tree branches in a chromosome is 5, the initial maximum tree depth is 3, the size of elite group is estimated at 5 (5% of the total population) and the maximum number of GP generations is estimated at 200. Figure 7.9 illustrates the evolution of the GP in terms of RMSE for the case of 3 variables selected from the SGNO rank table.

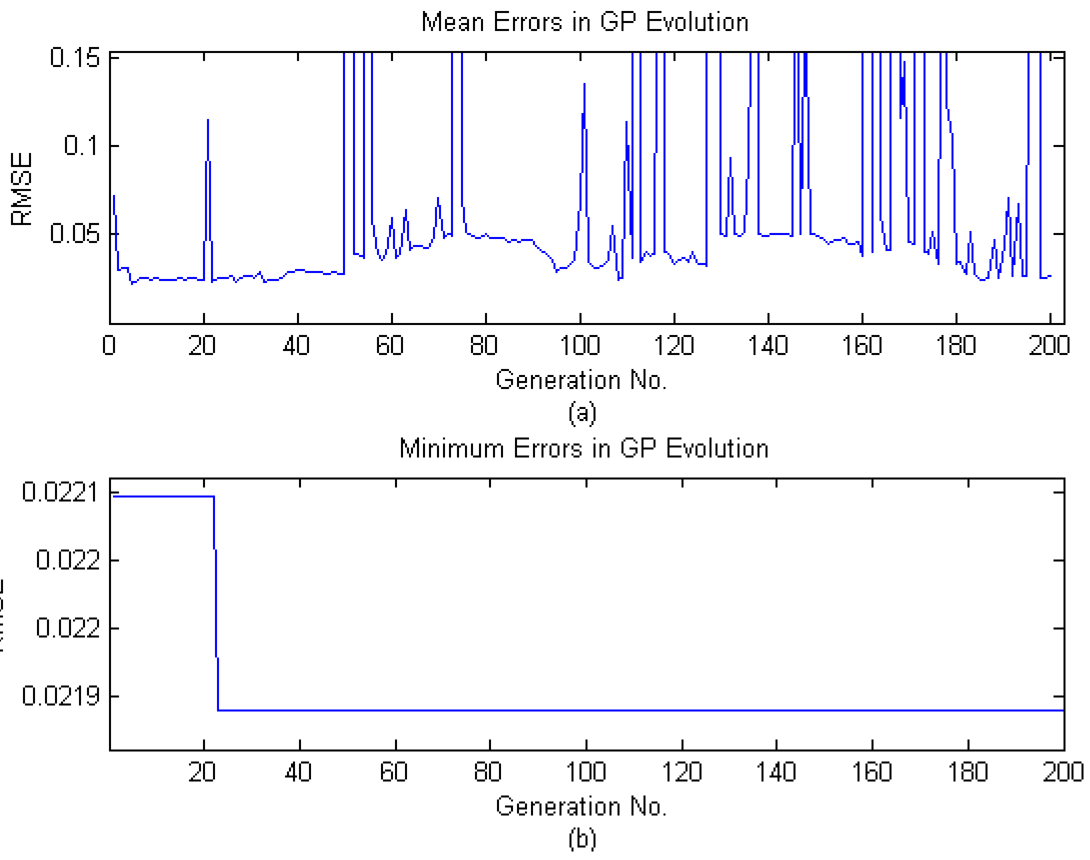

Figure 7.9: Evolution of GP for 3 variables. (a) mean errors in generations, (b) minimum errors in generations

As shown in Figure 7.9(a), the mean GP generation errors fluctuates vigorously and there are many generations producing extremely large mean errors due to the existences of certain chromosomes generating huge regression errors. Figure 7.9(b) illustrates the minimum error in each GP generation and a clear decline is present at the 21st generation, which indicates that the best performing chromosome is produced in that generation and survives throughout the following evolutions. This best performing chromosome generates a regression error (RMSE) at 0.0219 and the chromosome's expression is:

$$y = A * (x_1)^{-2/3} * \frac{x_1}{\sqrt{x_3}} + B * x_1 + C * (x_1)^{-2/3} * \frac{x_1}{x_2(x_2 + x_3)} + D$$

Table 7.6 lists the performance of the GP process and the discovered symbolic expressions using 6 and 9 variables from the SGNO rank table. It can be seen that as the number of variables increases, the symbolic expressions discovered by MBE-GP produce better results and these results are competitive to the performance of the 'optimal' NN models, which were 0.0241, 0.0116 and 0.0116 for the cases of 3, 6 and 9 input variables respectively.

Table 7.7: Symbolic expressions discovered by GP for various numbers of variables

No. of Variables	RMSE	Symbolic Expression
6	0.0127	$y = A * x_5 + B * (x_5 + x_6) * \left(\sqrt{x_6} - (x_5)^2\right) + C * \left(\sqrt{x_3} + x_4x_5\right)^4 + D * \mathrm{LOG}\left((x_2)^{-\frac{5}{2}}\right) + E * \sqrt{x_1 + x_3} + F$
9	0.0111	$y = A * \mathrm{LOG}\left(\sqrt{x_9x_5 + x_6 - x_5}\right) + B * \mathrm{LOG}(x_5) + C * \left(\frac{x_9 + x_4}{\sqrt{x_2}} * (x_3 - x_2 - x_1 + x_9) - \frac{x_9x_3x_4}{x_2}\right) + D$

7.8 Benchmarking with other Feature Selection (FS) Techniques

In this section, the performance of SGNO is benchmarked with four dimensionality reduction techniques, including PCA, FFS, BFS and GNMM.

7.8.1 Principal Component Analysis (PCA)

PCA reduces data dimensionality by transforming the original dataset linearly in to a new coordinate system (Dunteman, 1989). The detailed procedures of PCA were discussed in section 3.3.1.

In this application, PCA is used to transform the 50 input parameters and the performance of the new parameters (PCs) is evaluated using NNs. Figure 7.10 illustrates the variances carried in each PC. The figure displays only 4 PCs and the first PC carries the most variance, about 96%, while the other PCs carry only tiny amounts of variances. The accumulated variance plot shows that the first 4 PCs displayed explain variance close to 100%.

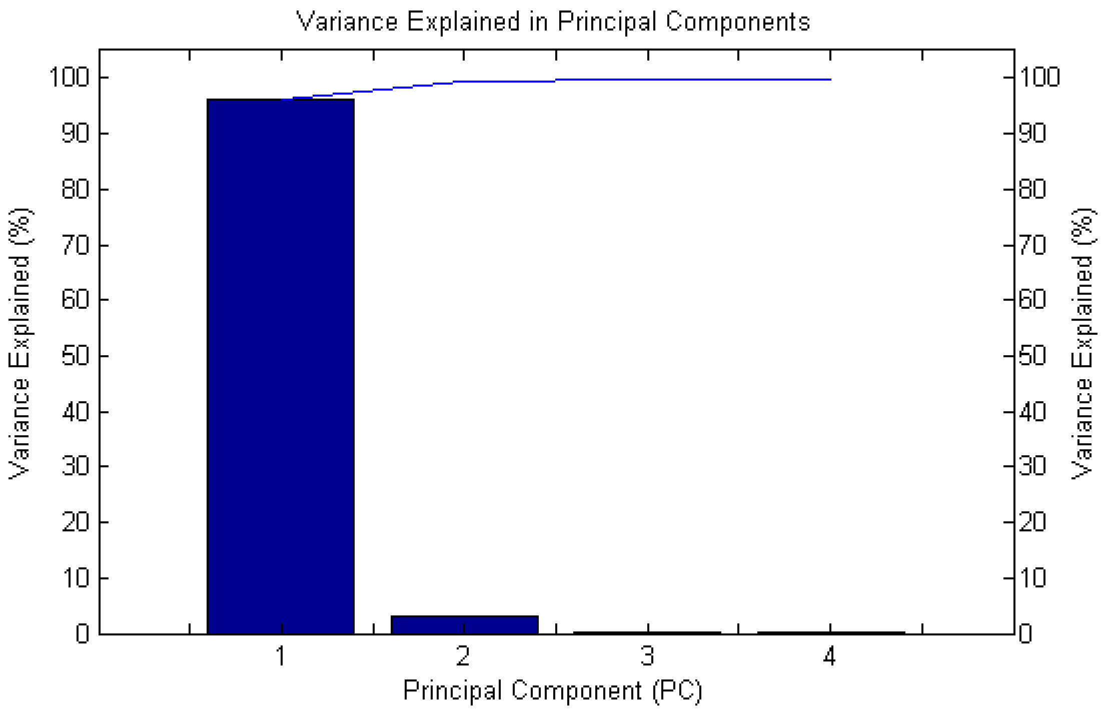

Figure 7.10: Variance explained in principal components

After the PCA transformation, NNs are used to evaluate the performance of the PCs by taking various numbers of PCs. Again, five-fold cross-validation and early-stopping training is employed to support the NN evaluation. The NN models are of single hidden processing layer and the number of hidden neurons is estimated by halving the number of inputs and output. Table 7.8 lists the performance of NN models using various numbers of PCs as input variables. Figure 7.11 illustrates the performance of these NN models against the same number of parameters picked from the variable rank table constructed in SGNO.

Table 7.8: PCA re-evaluation results

Size	1	2	3	4	5	6	7	8	9
RMSE	0.0384	0.0362	0.0248	0.0241	0.0181	0.0185	0.0191	0.0181	0.0164

Size	10	11	12	13	14	15	16	17	18
RMSE	0.0136	0.0142	0.0132	0.0139	0.0143	0.0143	0.0137	0.0145	0.0145

Size	19	20	21	22	23	24	25	26	27
RMSE	0.0131	0.0135	0.0136	0.0135	0.0131	0.0137	0.0138	0.0132	0.013

Size	28	29	30	31	32	33	34	35	36
RMSE	0.014	0.0129	0.0139	0.0129	0.0128	0.0133	0.0123	0.012	0.0113

Size	37	38	39	40	41	42	43	44	45
RMSE	0.0121	0.0113	0.0119	0.0111	0.0125	0.013	0.0125	0.0141	0.0167

Size	46	47	48	49	50
RMSE	0.0152	0.019	0.0189	0.0187	0.0178

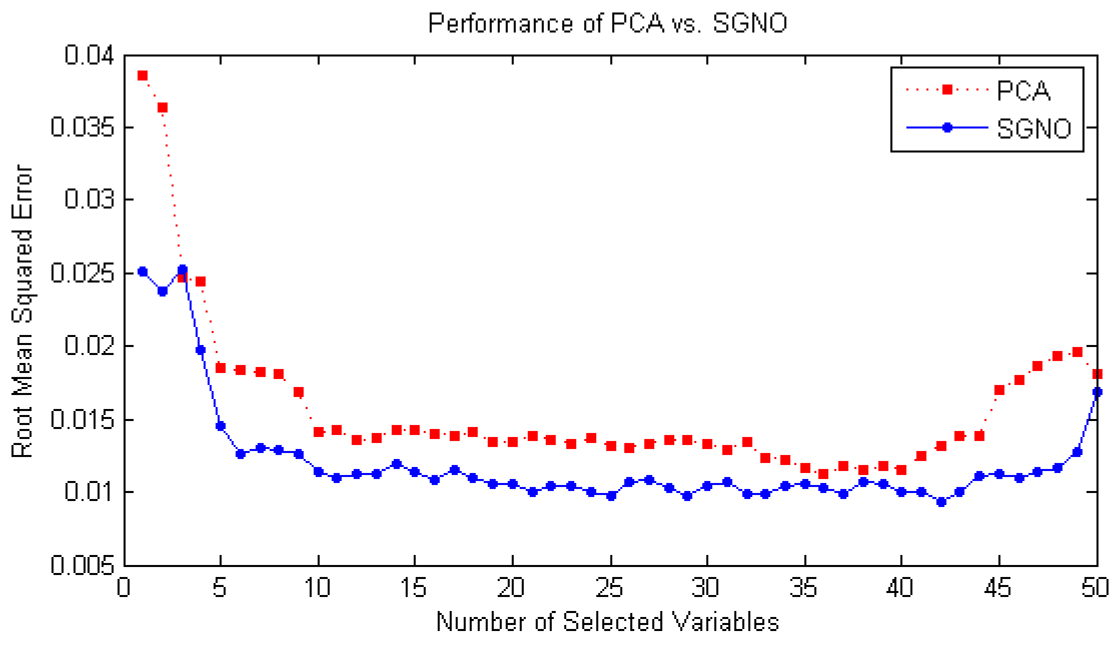

Figure 7.11: Performance of NN models using PCA against SGNO

As shown in Figure 7.11, both performance plots are of similar patterns, a rapidly reducing start followed by a relatively steady plot fluctuating around a converged value, and ends with a clear rise. In addition, the parameters selected by SGNO produce better performance compared with the PCA as the RMSEs generated using the SGNO selected parameters are always lower than those generated using PCA, and the largest difference is up to the value of 0.0133 when only one input variable is used in the NN models.

7.8.2 Forward Feature Selection (FFS) and Backward Feature Selection (BFS)

FFS and BFS are conventional sequential FS techniques. The detailed procedures are discussed in section 3.3.2. In this application, FFS and BFS only set out the structure of the FS routine and the evaluation of the selected variables is performed using the NN models. The NN models used in this application are of single hidden processing layer and the number of hidden neurons is estimated by halving the number of inputs and output. Again, five-fold cross-validation and early-stopping are employed.

The following lists present the order of variables added to the selection list in FFS and the reversed order of variables removed from the selection in BFS. Hence, the variables are in the descending order of preferences in both lists.

FFS: [3, 40, 5, 6, 17, 31, 49, 10, 7, 22, 39, 42, 29, 25, 12, 37, 1, 33, 15, 13, 44, 16, 8, 32, 11, 26, 30, 38, 14, 34, 50, 4, 9, 21, 35, 23, 18, 24, 27, 48, 20, 47, 36, 19, 2, 28, 46, 41, 43, 45]

BFS: [3, 39, 10, 6, 50, 15, 49, 4, 24, 5, 18, 35, 26, 31, 16, 48, 9, 36, 8, 43, 28, 30, 27, 25, 13, 46, 21, 34, 7, 32, 47, 19, 44, 22, 29, 17, 14, 23, 12, 41, 11, 42, 1, 40, 2, 38, 33, 45, 37, 20]

Comparing with the variable importance rank table constructed in SGNO, which is **[3, 20, 30, 5, 40, 39, 10, 33, 13, 6, 16, 17, 19, 18, 36, 9, 38, 26, 1, 15, 23, 29, 24, 28, 8, 21, 37, 14, 49, 27, 22, 34, 35, 11, 50, 31, 12, 25, 4, 2, 43, 32, 7, 44, 46, 42, 48, 47, 41, 45]**, the parameters are ordered in distinct ways. However, they all identify the parameters No.3 as the most preferable parameter and the parameter No.45 is less favourable than most of the variables. Table 7.9 lists the performance of the preference lists produced by BFS and FFS using NN models with various numbers of input variables selected from the preference lists.

Table 7.9: BFS and FFS re-evaluation results

Size	1	2	3	4	5	6	7	8	9
BFS	0.025	0.0174	0.0178	0.0144	0.0139	0.0134	0.0138	0.0138	0.0141
FFS	0.025	0.0166	0.0143	0.0133	0.0128	0.0126	0.0136	0.0128	0.0127

Size	10	11	12	13	14	15	16	17	18
BFS	0.0117	0.0124	0.0119	0.0116	0.0112	0.0115	0.0117	0.0116	0.0115
FFS	0.013	0.012	0.0118	0.0117	0.0118	0.012	0.0115	0.0111	0.011

Size	19	20	21	22	23	24	25	26	27
BFS	0.0115	0.0113	0.0118	0.011	0.0117	0.0114	0.012	0.0114	0.0122
FFS	0.0106	0.0106	0.0116	0.0114	0.0111	0.011	0.0114	0.0117	0.0106

Size	28	29	30	31	32	33	34	35	36
BFS	0.0122	0.0115	0.0109	0.011	0.0112	0.0119	0.0131	0.0113	0.0132
FFS	0.0115	0.0112	0.0109	0.0111	0.0108	0.0109	0.0111	0.0101	0.0109

Size	37	38	39	40	41	42	43	44	45
BFS	0.0124	0.0128	0.0115	0.0134	0.0133	0.0128	0.0138	0.0137	0.0134
FFS	0.011	0.0108	0.0107	0.0119	0.0117	0.0115	0.0121	0.0117	0.0114

Size	46	47	48	49	50
BFS	0.0137	0.0136	0.0167	0.016	0.0162
FFS	0.0114	0.0122	0.013	0.0131	0.0169

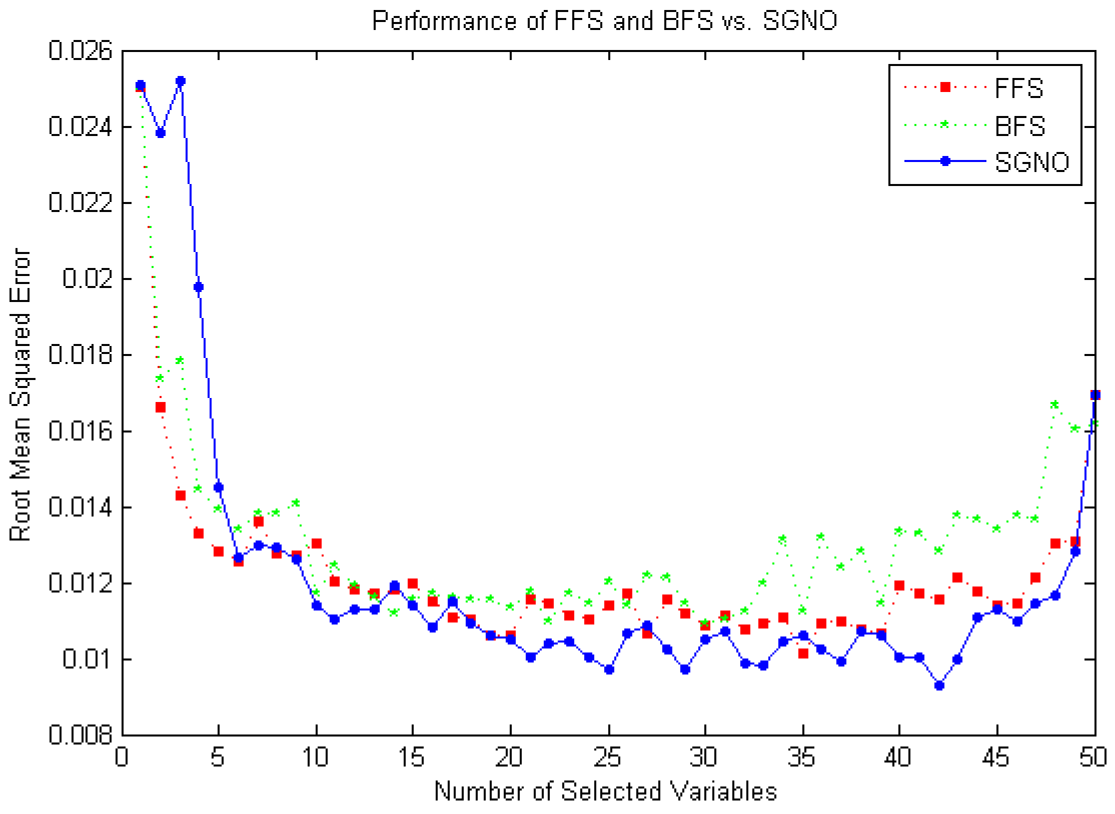

Figure 7.12: Performance of NN models using BFS and FFS against SGNO

Figure 7.12 illustrates the performance of BFS and FFS against SGNO using various numbers of variables from the preference lists. Although BFS and FFS produced different parameter preference lists, the performance plots are of similar patterns, a rapidly decreasing curve at the beginning followed by a gradually decreasing pattern with high fluctuations and a rapidly increasing tail. The FFS produces lower RMSEs in general, especially at the tail of the performance plot. Comparing with BFS and FFS, the SGNO is not performing well when the number of selected variables is small (less than 5). When the number of selected variables becomes greater than 5, the SGNO variable rank table starts generating results competitive to BFS and FFS. When the number of variables is greater than 20, the SGNO produces better performance than BFS and FFS with noticeable advantages as illustrated in Figure 7.12.

7.8.3 Genetic Neural Mathematical Method (GNMM)

GNMM is a hybrid data mining technique incorporating GA and ANNs that performs FS based on the appearance frequencies of the variables in the GA process. The variables of higher appearance percentages are considered more favourable than those of lower appearance percentages (Yang, 2010; Yang et al., 2007). The detailed discussion of GNMM is available in section 3.3.3. Figure 7.13 illustrates the appearance percentages of the variables discovered in GNMM process.

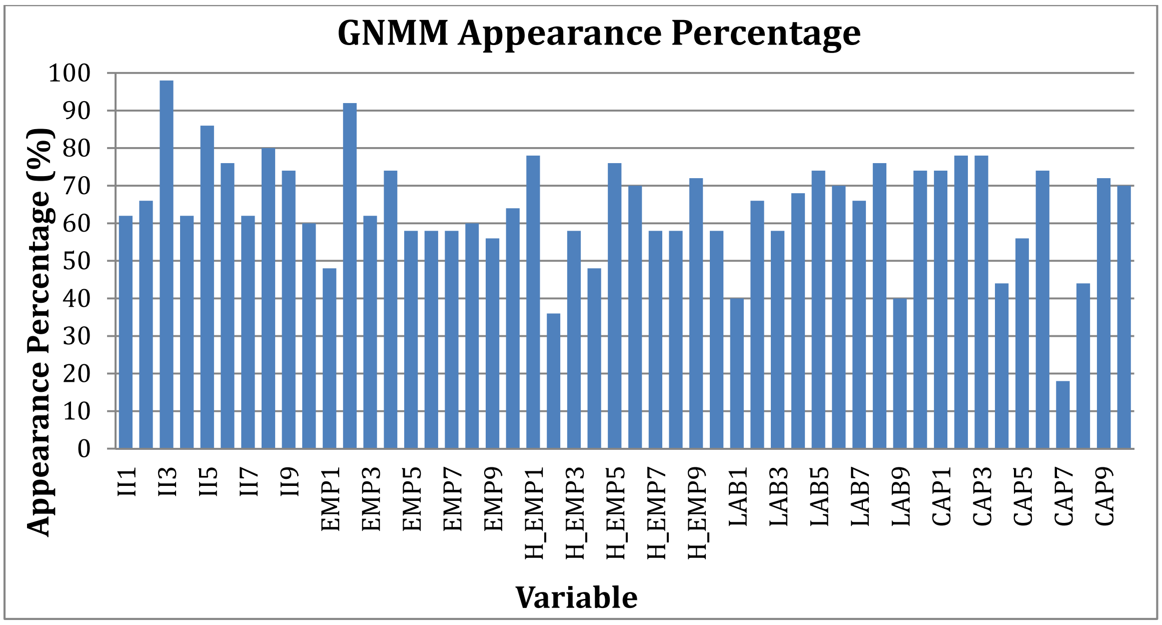

Figure 7.13: Variable appearance percentages in GNMM

As shown in Figure 7.13, a large number of the input variables are of high appearance percentages (over 60%). Among all the parameters, II_3, EMP_2 are of the highest appearances (over 90%), followed by II_5 and II_8 (over 80%). The following list contains the ordered variables based on their appearance percentages in descending order.

[II_3, EMP_2, II_5, II_8, H_EMP_1, CAP_2, CAP_3, II_6, H_EMP_5, LAB_8, II_9, EMP_4, LAB_5, LAB_{10}, CAP_1, CAP_6, H_EMP_9, CAP_9, H_EMP_6, LAB_6, CAP_{10}, LAB_4, II_2, LAB_2, LAB_7, EMP_{10}, II_1, II_4, II_7, EMP_3, II_{10}, EMP_8, EMP_5, EMP_6, EMP_7, H_EMP_3, H_EMP_7, H_EMP_8, H_EMP_{10}, LAB_3, EMP_9, CAP_5, EMP_1, H_EMP_4, CAP_4, CAP_8, LAB_1, LAB_9, H_EMP_2, CAP_7]

The following index list contains the indices (positions) of the variables in GA chromosomes ordered based on the list above.

[3, 12, 5, 8, 21, 42, 43, 6, 25, 38, 9, 14, 35, 40, 41, 46, 29, 49, 26, 36, 50, 34, 2, 32, 37, 20, 1, 4, 7, 13, 10, 18, 15, 16, 17, 23, 27, 28, 30, 33, 19, 45, 11, 24, 44, 48, 31, 39, 22, 47]

Table 7.10 lists the performance of the rank list produced by GNMM using NN models with various numbers of input variables selected from the lists and Figure 7.14 illustrates the performance of the GNMM discovered parameter rank table against SGNO.

Table 7.10: GNMM re-evaluation results

Size	1	2	3	4	5	6	7	8	9
RMSE	0.0251	0.0257	0.0211	0.0211	0.0199	0.0203	0.02	0.0186	0.0181

Size	10	11	12	13	14	15	16	17	18
RMSE	0.0151	0.0141	0.0138	0.0135	0.0122	0.0134	0.0136	0.0135	0.0138

Size	19	20	21	22	23	24	25	26	27
RMSE	0.0131	0.0129	0.0128	0.0127	0.0123	0.0124	0.0123	0.0123	0.0117

Size	28	29	30	31	32	33	34	35	36
RMSE	0.0114	0.0117	0.0124	0.0113	0.0121	0.0113	0.0113	0.0116	0.0118

Size	37	38	39	40	41	42	43	44	45
RMSE	0.0115	0.0118	0.0117	0.0113	0.0112	0.0151	0.0144	0.0144	0.0141

Size	46	47	48	49	50
RMSE	0.0171	0.0161	0.0157	0.0162	0.0172

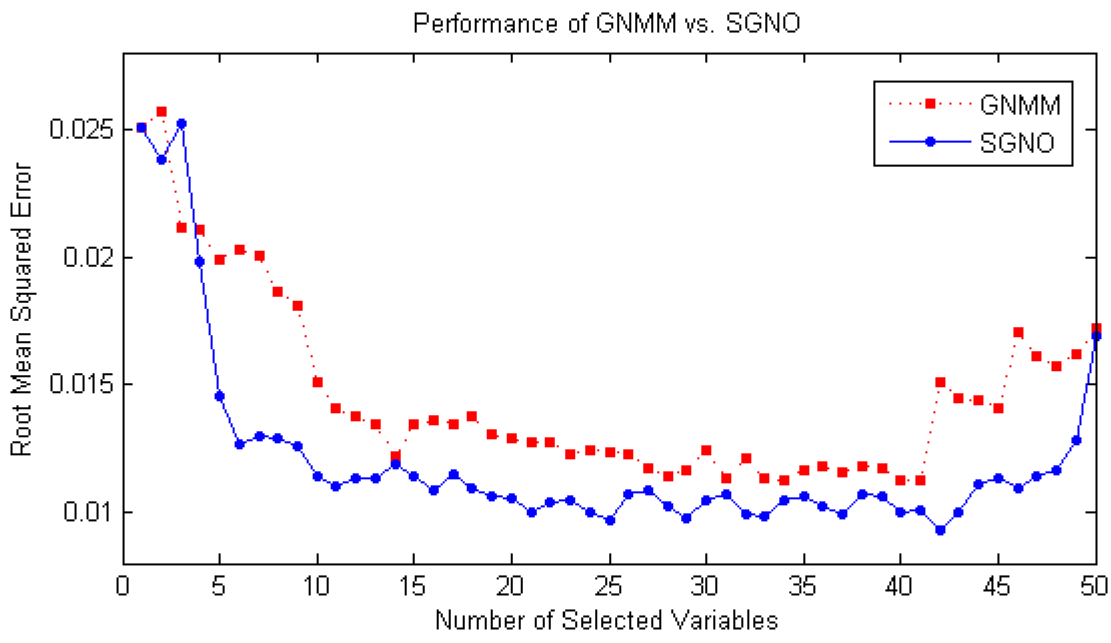

Figure 7.14: Performance of GNMM against SGNO

As shown in Figure 7.14, the SGNO's performance is generally better than GNMM as the SGNO's RMSEs are lower than those for GNMM. When the number of selected variables is less than 5, GNMM performs in a similar manner to SGNO. When the number of selected variables is between 5 and 10, SGNO shows significant advantages over GNMM and the largest difference is up to the value of 0.0077 (6 input variables). When the number of selected variables exceeds 10, the advantage of SGNO gradually declines until it reaches the tails of the performance plots.

7.8.4 Benchmarking summary

Figure 7.15 summarises the performance of the variable rank tables constructed using all the techniques presented. It shows that among these 5 techniques, PCA performs the worst in general, as the performance plot is always above those of the others. BFS, FFS, GNMM and SGNO are of distinct performance when the number of input variable is less than 5, where BFS and FFS are superior to GNMM and SGNO. When the number of input variables exceeds 5, BFS, FFS, GNMM and SGNO start producing similar RMSEs. The general patterns in these performance plots, which is a rapid drop followed by a fluctuation around a converged value with a clear rise at the tail, indicate that the first 5 to 10 variables in the variable rank tables generated by these techniques are sufficient to estimate the gross production growth with an acceptable level of accuracy using NN models. Furthermore, BFS, FFS, GNMM and SGNO all successfully identified the five most important variables from the original data.

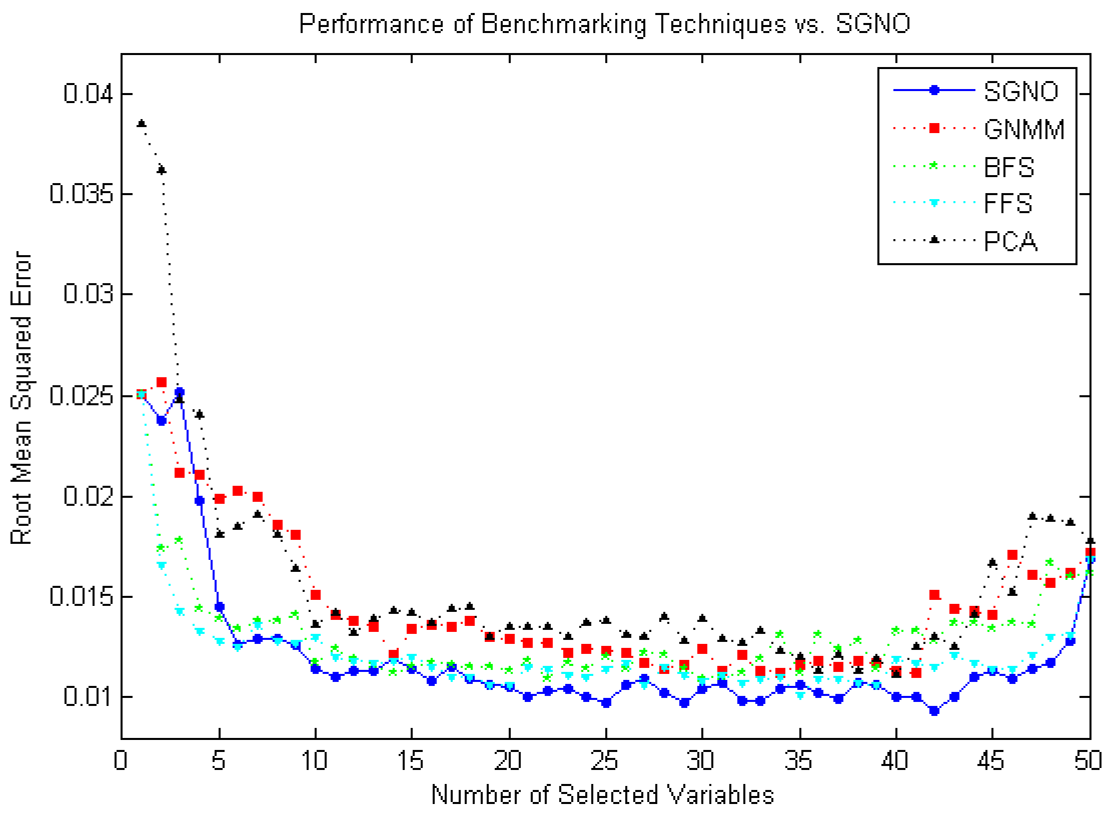

Figure 7.15: Performance of all benchmarking techniques against SGNO

7.9 Conclusion

In this chapter, SGNO is used to establish NN models to predict the growth in industrial productions using five types of economic inputs in ten industrial sectors. The data source is an economic database created in the EU KLEMS project to store the detailed measures of industry output, input and growth at industry level. The database covers 30 countries from 1970 onwards. After establishing prediction models, the symbolic relationship between the model inputs and output are discovered using the MBE-GP. In addition, the performance of SGNO is benchmarked against three conventional FS techniques (PCA, BFS and FFS) and a recently developed optimisation algorithm (GNMM).

An analysis of the results shows that only five to ten variables effectively contribute to the output in NN models, most of the residual variables do not contribute significantly to the NN models and some of the variables have a negative effect (adding noises and thus increasing output errors) in the NN models (see Figure 7.7). The influence/importance rank table constructed by SGNO is benchmarked with four FS techniques, including BFS, FFS, GNMM and PCA. Among all these techniques, PCA always generates the highest RMSEs and thus gives the worst performance. When the number of selected variables is less than five, BFS and FFS are superior to GNMM and SGNO. However, when the number of selected variables exceeds five, the RMSEs produced by BFS, FFS and SGNO are of minor differences and significantly less than the RMSEs generated by GNMM and PCA. Generally speaking, SGNO is competitive with these benchmarking techniques.

After identifying the influential levels of all the variables, MBE-GP successfully extracted concise symbolic relationships between various numbers of input variables and the corresponding outputs. The regression errors produced using these symbolic relationships are competitive with those generated by NN models.

References

Ahmed F.E. (2005) Artificial neural networks for disgnosis and survival prediction in colon cancer. Molecular Cancer 4.

Boritz J.E., Kennedy D.B., de Mirandae Albuquerque A. (1995) Predicting corporate failure using a neural network approach. Intelligent Systems in Accounting, Finance and Management 4:95-111.

Dunteman G.H. (1989) Principal Components Analysis SAGE Publications.

Duzgun R. (2010) Generalized Regression Neural Networks for Inflation Forecasting. International Research Journal of Finance and Economics 51:59-70.

Herbrich R., Keilbach M., Graepel T., Obermayer K. (1999) Neural networks in economics: background, applications, and new developments. Advances in Computational Economics 11:169-196.

Junoh M.Z.H.M. (2004) Predicting GDP growth in Malaysia using knowledge-based economy indicators: a comparison between neural network and economitric approaches. Sunway College Journal 1:39-50.

Kaastra I., Boyd M. (1996) Designing a neural network for forecasting financial and economic time series. Neurocomputing 10:215-236.

Koszerek D., Havik K., Mc Morrow K., Roger W., Schonborn F. (2007) European Economy - Economic Papers: An overview of the EU KLEMS Growth and Productivity Accounts, European Commission. pp. 107.

Moody J. (1995) Economic Forecasting: Challenges and Neural Network Solutions, The International Symposium on Artificial Neural Networks, Hsinchu, Taiwan.

O'Mahony M., Timmer M.P. (2009) Output, Input and Productivity Measures at the Industry Level: the EU KLEMS Database. The Economic Journal 119:374-403.

Odom M.D., Sharda R. (1990) A Neural Network Model for Bankruptcy Prediction, IEEE International Conference on Neural Networks, San Diego.

Tam K.Y., Kiang Y.M. (1992) Managerial Application of Neural Networks: The Case of Bank Failure Predictions. Management Science 38:926-947.

Yang J.H. (2010) Intelligent data mining using artificial neural networks and genetic algorithms: techniques and applications, Engineering, University of Warwick, Coventry. pp. 260.

Yang J.H., Hines E.L., Iliescu D.D., Leeson M.S. (2007) GNMM and accurate longitudinal dispersion coefficient prediction, The Seventh UK Chinese Association of Resource and Environment (CARE) Annual Meeting, Greenwich, London.

Zhang G., Hu M.Y., Patuwo B.E., Indro D.C. (1999) Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis. European Journal of Operational Research 116:16-32.

CHAPTER 8

Conclusions and Further Work

8.1 Overview

This chapter summarises the main outcomes of this research and presents an outline of the main results followed by suggestions for possible future research.

8.2 Main Research Findings

8.2.1 SGNO Briefing

In this thesis, a novel hybrid IST technique is introduced, which aims to reduce data dimensionality (Feature Selection, FS) by identifying the importance of variables in the system and thus suggests the group of variables that meets users' requirements. The feasibility of SGNO is evaluated with four practical applications in three novel research areas, including plant science, civil engineering and economics. The performance of SGNO is benchmarked with three conventional FS techniques and a recently developed IST based technique, these include PCA, FFS, BFS and GNMM. The main contributions of SGNO include:

- Establishing neural models between input variables and output variable;
- Identifying the importance of variables in the neural models and thus producing a variable importance ranking table;

- Determining the possible outcomes of various groups of variables (less number of variables) and amount of information preserved from the original data set;
- Discovering symbolic expression of the relationship between input and output variables that may represent the neural models.

SGNO consists of three key modules, which are the GA module, the NN module and the SA module (see section 3.2). The key functionalities of these modules and their interactions are summarised as follows:

- The GA module determines the general structure of this optimisation algorithm. It optimises the input variables by evolving various combinations of inputs using the GA operators. The fitness or performance of each proposed combination of inputs, represented using binary chromosomes, was determined using the NN module.
- The NN module determines the performance of the proposed input combination by constructing a neural regression model using MLP.
- The SA module provides a refined analysis on a group of input combinations of good performance. It investigates how the NN models respond to the variations in its inputs and thus identifies the influence (importance) of each variable on the model. The global influence of each individual input variable is derived by averaging their influences in all the selected combinations to determine their importance.
- Based on the variables' global influences generated in the SA module, variables of high influence can be selected as the representative of the

original data (the exact number of selected variables can be determined by the users).

Theoretical complexity analysis showed that among SGNO and the benchmarking techniques, PCA is of linear processing complexity based on the total number of variables; BFS and FFS require quadratic processing time; GNMM and SGNO are of more complicated cases, where the processing time is dependent not only on the number of variables but also on the predefined size of GA population (see section 3.4).

8.2.2 Case Study Results

In this thesis, a total of four datasets (chapters 4 to 7) have been used to demonstrate implementation and the performance of SGNO in three different novel application scenarios.

In chapter 4, SGNO is used to construct NN models which are capable of making short-term predictions of greenhouse tomato yield based on the environmental conditions and yields in previous weeks. The results show that radiation is the most important variable group overall, while the VPD is the least important. Depending on the number of variables selected from the SGNO results, re-evaluation shows that the established models produce average errors in the range of 0.09 and 0.14. The SGNO's evaluation results can be found in section 4.6. Compared with the benchmarking techniques, SGNO always produces lower regression errors than PCA and GNMM. In the case of BFS and FFS, SGNO provides comparable performance when the number of selected variables is less than 10, and becomes superior thereafter (see section 4.8.4 and figure 4.20 for details).

In chapter 5, SGNO is used to estimate the longitudinal dispersion coefficient in natural rivers using the hydraulic and geometrical parameters collected at various locations of a river. The result shows that among the 20 evaluated variables, only half of them (approximately 10 to 12) contribute significantly to the regression models. Re-evaluation shows that the performance, in terms of RMSEs, of SGNO is in the range of 0.45 and 0.65 depending on the number of variables. The evaluation results can be found in section 5.6. Benchmarking results show that SGNO produces performance competitive to BFS and FFS with minor advantages in many situations, especially when the number of variables is between 5 and 12, while GNMM and PCA are not performing well in this application (see section 5.8.4 and figure 5.15 for details).

In chapter 6, SGNO's performance is evaluated when it is applied identify the key features that determine the amount of discharge occurring at coastal structures. SGNO identifies the crest freeboard and the roughness of the structure as the most influential factors in overtopping. The performance of SGNO is in the range of 0.47 and 1.01 (see section 6.6 for details). Compared with the benchmarking techniques, SGNO performs better than GNMM and PCA on average but not as well as BFS and FFS (see section 6.8.4 and figure 6.19 for details).

In chapter 7, SGNO is applied to process some economic data. SGNO is used to model the total industry output growth in terms of the growths in industry inputs, including intermediate inputs (materials, energy, etc.), engagement of persons, working hours, labour and capital compensations. The results produced by SGNO show that only approximately 5 to 10 variables effectively contribute to the output and some of the variables have negative effects (see section 7.6 for details). Comparing with the benchmarking techniques, SGNO is generally performs the best when the number of variables is greater than 5. When the number of variables is less than 5, BFS and FFS generate the lowest RMSEs (see section 7.8.4 and figure 7.15 for details).

Actual processing time of SGNO showed that the computation time is not only dependent on the number of variables and the size of the dataset but also on the complexity of the problem, which determines how quickly NNs can learn from the dataset (see section 3.5).

8.3 Weaknesses of SGNO and Further Works

The current implementation of SGNO only produces the evaluation results of the importance ranking tables and the final judgement of the appropriate number of features to retain is left to the users. Further improvement of SGNO may have an automatic function to generate recommendations of appropriate groups of features. Possible criteria to judge may include:

- Increasing the number of variables cannot improve the performance of NN models significantly;
- Increasing the number of variables makes NN models produce worse results.

The current implementation of SGNO employs GA with a relatively large population size (5 times the number of all features). Large population size generally makes GA more time consuming. A possible improvement is to reduce the population size of GA to make the algorithm more time efficient. In addition, the SA module in SGNO is used more like an additional package that carries out refined explorations in the search space already discovered by the GA. A possible improvement in the algorithm is to increase the efficiency of the GA by adopting SA into its operations, such as initialisation, crossover and mutation (Chen et al., 2008).

The MBE-GP, which is used in this thesis to find the symbolic relationships between input and output variables, introduced the idea of bringing coefficients into the branches of the tree structured solutions. A possible improvement is assigning coefficients to all elements (branches and nodes) in the tree to make the tree more flexible. The coefficients may be initialised using random numbers and evolve to good fitting numbers using learning algorithms, such as back propagation.

It is obvious that SGNO is not working very well when a small number of variables (less than 5), BFS and FFS generally produce better results under these cases (see figures 4.20, 6.19 and 7.15). This is due to the fact that the initialisation operation of GA cannot start with a small number of variables to avoid zero selection and GA evolves rapidly in the first few generations. Thus the exploration in low dimensional space is not sufficient. Further development may enhance the exploration in low dimensional space by slowing down the evolution.

As SGNO is a general purpose IST tool and this thesis only presents the applications of SGNO in data regression, further works may include the implementation of SGNO in other types of applications, such as rule extraction and classification. Another possible direction for future research is to explore the feasibility of SA in terms of assisting other types of IST techniques, such as GPs, FLS, NNs, etc (Vladislavleva et al., 2010; Yeung et al., 2010).

References

Chen Y., Zhang Y., Wei Y. (2008) Application of improved genetic algorithm combining sensitivity analysis to reactive power optimization for power system, Electric Utility Deregulation and Restructuring and Power Technologies, 2008, Nanjing. pp. 798-803.

Vladislavleva K., Veeramachaneni K., Burland M., Parcon J., O'Reilly U.M. (2010) Knowledge mining with genetic programming methods for variable selection in flavor design, The 12th annual conference on Genetic and evolutionary computation conference, Portland, OR, USA. pp. 941-948.

Yeung D.S., Cloete I., Shi D., Ng W.W.Y. (2010) Sensitivity Analysis for Neural Networks Springer.

Appendix A. Matlab Program for SGNO

A.1 Neural Network Module

```
function out = mlpeval(chrom)
%   CHROM,  chromosome to be evaluated
%   OUT,    evaluation result

global A_p  A_t  B_p  B_t  C_p  C_t  D_p  D_t  E_p  E_t
%   five-fold evaluation groups
global chroms
global chromfitness
global chromnets
global popsize gencount

sel = find(chrom==1);

%   extract selected variables from the dataset
ap = A_p(sel, :);
at = A_t;
bp = B_p(sel, :);
bt = B_t;
cp = C_p(sel, :);
ct = C_t;
dp = D_p(sel, :);
dt = D_t;
ep = E_p(sel, :);
et = E_t;

errors = [];
nets = {};

for i = 1:5
    repno = 20;
    repmse = 0;
    for rep = 1:repno
        switch i
            case 1
                setP = {[ap, bp, cp], dp, ep};
                setT = {[at, bt, ct], dt, et};
                mlpnet = newff(ap, at, ceil((1+sum(chrom))/2));
            case 2
                setP = {[ep, ap, bp], cp, dp};
                setT = {[et, at, bt], ct, dt};
                mlpnet = newff(ep, et, ceil((1+sum(chrom))/2));
            case 3
                setP = {[dp, ep, ap], bp, cp};
                setT = {[dt, et, at], bt, ct};
                mlpnet = newff(dp, dt, ceil((1+sum(chrom))/2));
            case 4
                setP = {[cp, dp, ep], ap, bp};
                setT = {[ct, dt, et], at, bt};
                mlpnet = newff(cp, ct, ceil((1+sum(chrom))/2));
            case 5
                setP = {[bp, cp, dp], ep, ap};
                setT = {[bt, ct, dt], et, at};
                mlpnet = newff(bp, bt, ceil((1+sum(chrom))/2));
        end

        mlpnet.trainFcn = 'trainscg';
        mlpnet.trainParam.showWindow = 0;
        mlpnet = mytrainnew(mlpnet, setP, setT);

        switch i
            case 1
                simr = sim(mlpnet, ep);
                mse = mean((simr-et).^2);
            case 2
                simr = sim(mlpnet, dp);
                mse = mean((simr-dt).^2);
            case 3
                simr = sim(mlpnet, cp);
                mse = mean((simr-ct).^2);
            case 4
                simr = sim(mlpnet, bp);
                mse = mean((simr-bt).^2);
            case 5
                simr = sim(mlpnet, ap);
                mse = mean((simr-at).^2);
        end

        repmse = repmse+mse;
    end
    mse = repmse/repno;
    errors = [errors, mse];
    nets = [nets, mlpnet];
end

chroms = [chroms; chrom];
chromfitness = [chromfitness; errors];
chromnets = [chromnets; nets];

% save evaluation results at the end of each generation
netlength = size(chroms, 1);
fprintf('Generation: %d, \t Chromosome: %d \n', gencount, netlength)

if mod(netlength, popsize)==0
    save(['gen', num2str(gencount), '_chroms.mat'], 'chroms');
    chroms = [];
    save(['gen', num2str(gencount), '_chromfitness.mat'],
'chromfitness');
    chromfitness = [];
    save(['gen', num2str(gencount), '_chromnets.mat'], 'chromnets');
    chromnets = {};
    gencount = gencount + 1;
end

out = mean(errors);
fprintf('Fitness: %d, Chrom Size: %d\n\n', out, sum(chrom));
```

A.2 Sensitivity Analysis

```
function out = sensitivity(chrom, nets)
```

```
%   CHROM,  binary chromosome to be tested
%   NETS,   trained MLPs associated with the CHROM
%   OUT,    sensitivity scores of all selected variables represented
%           by the ON bits in CHROM

inds = find(chrom==1);
out = zeros(1, length(chrom));

for i = 1:length(inds)
    sens = [];
    for j = 1:5
        sens(j) = montecarlo(nets(j), chrom, i);
    end
    out(inds(i)) = mean(sens);
end
```

A.3 Monte Carlo Simulation

```
function out = montecarlo (netin, chrom, ind)
%   NETIN,  instance of trained MLP
%   CHROM,  binary chromosome to be tested
%   IND,    index of the variable to be tested
%   OUT,    sensitivity score of a particular ON bit in CHROM
%           measured using a trained MLP, NETIN

global chmax chmin
%   upper and lower boundaries of all variables

errors = [];
errm = [];
convergestep = 25;
similarity = 0.99;

chromsize = sum(chrom);
ids = find(chrom);
runsize = 500;

zftemp = rand(chromsize+1, runsize);
randin = zftemp;
for rs = 1:runsize
   randin(1:chromsize, rs) = zftemp(1:chromsize, rs).*(chmax(ids)-
chmin(ids))+chmin(ids);
   randin(end, rs) = zftemp(end, rs)*(chmax(ids(ind))-
chmin(ids(ind)))+chmin(ids(ind));
end

y1 = sim(netin, randin(1:chromsize, :));
randscale = abs(randin(ind,:)-randin(end,:));
randin(ind,:) = randin(end,:);
y2 = sim(netin, randin(1:chromsize, :));

errors = abs((y1-y2)./randscale);
ttt = find(randscale==0);
errors(ttt) = 0;

for i = 1:runsize
    errm(i) = mean(errors(1:i));
```

```
end

while true
    zftemp = rand(chromsize+1, runsize);
    randin = zftemp;
    for rs = 1:runsize
       randin(1:chromsize, rs) = zftemp(1:chromsize,
rs).*(chmax(ids)-chmin(ids))+chmin(ids);
       randin(end, rs) = zftemp(end, rs)*(chmax(ids(ind))-
chmin(ids(ind)))+chmin(ids(ind));
    end

    y1 = sim(netin, randin(1:chromsize, :));
    randscale = abs(randin(ind,:)-randin(end,:));
    randin(ind,:) = randin(end,:);
    y2 = sim(netin, randin(1:chromsize, :));

    err = abs((y1-y2)./randscale);
    ttt = find(randscale==0);
    err(ttt) = 0;
    errors = [errors, err];

    for i = length(errm)+1:length(errm)+runsize
       errm(i) = mean(errors(1:i));
       te = errm(i-convergestep+1:i)./errm(i-convergestep:i-1);

       nantest = isnan(te);
       if (sum(nantest)>0)
           out = mean(errors(1:i));
           return
       end

       if sum(te)>=similarity*convergestep && sum(te)<=convergestep
          out = errm(i);
          return
       end
    end
end
```

Appendix B. Matlab Program for MBE-GP

B.1 Branch (Tree) Generation

```
function treeout = maketree(depth)

global operatornames operatorarity operatorsize
global terminalnames terminalsize
global maxpower
% type,         0->terminal 1->operator
% operator,     id(index) in operatornames
% operand,      subtrees
% addition,     additional info for power function

if rand<=1/5
    treeout.type = 0;
else
    treeout.type = 1;
end

if depth==1
    treeout.type = 0;
    treeout.operand = randi(terminalsize);
    return
end

if treeout.type==0
    treeout.type = 0;
    treeout.operand = randi(terminalsize);
else
    treeout.operator = randi(operatorsize);

    if strcmp(operatornames{treeout.operator}, 'POW')
        switch randi(3)
            case 1
                treeout.addition = randi([2, maxpower]);
            case 2
                treeout.addition = -randi([2, maxpower]);
            case 3
                treeout.addition = 1/randi([2, maxpower]);
        end
    end

    switch operatorarity(treeout.operator)
        case 1
            treeout.operand = maketree(depth-1);
        case 2
            treeout.operand = {maketree(depth-1), maketree(depth-1)};
    end
end
```

B.2 Tree Evaluation

```
function out = evaltree(treestruct, interms)
```

```
global operatornames operatorfuns operatorarity

if treestruct.type==0
    out = interms(treestruct.operand);
else
    switch operatorarity(treestruct.operator)
        case 1
            if strcmp(operatornames{treestruct.operator}, 'POW')
                out = feval(operatorfuns{treestruct.operator},
evaltree(treestruct.operand, interms), treestruct.addition);
            else
                out = feval(operatorfuns{treestruct.operator},
evaltree(treestruct.operand, interms));
            end
        case 2
            out1 = evaltree(treestruct.operand{1}, interms);
            out2 = evaltree(treestruct.operand{2}, interms);
            out = feval(operatorfuns{treestruct.operator}, out1,
out2);
    end
end
```

B.3 Chromosome Evaluation

```
function [out, teval] = trainchrom(in, p, t)
% calculate the regression coefficients of the chromsome
% in:   chromosome structure
% p:    nxm matrix, m terminals and n records
% t:    nx1 matrix
% out:  output chromosome with regression coefficients

global terminalsize

if size(p,2)~=terminalsize
    error('input p must match terminal size')
end

out = in;
teval = ones(size(p, 1), 1+in.numofchrom);

for i = 1:size(p,1)
    for j = 1:in.numofchrom
        teval(i,j) = evaltree(in.chroms{j}, p(i,:));
    end
end

try
    out.coeffs = regress(t, teval);
catch exception
    out.coeffs = inf * ones(1+in.numofchrom, 1);
    disp('Training exception raised!!!')
end
```

```
function [out, teval] = simchrom(in, p)
% simulate chrom with certain terminals (p)
% p: nxm matrix, m terminals and n records
% out: nx1 matrix

global terminalsize

if isempty(in.coeffs)
    error('coefficient array must not be empty')
end

if size(p,2)~=terminalsize
    error('input p must match terminal size')
end

teval = ones(size(p, 1), 1+in.numofchrom);

for i = 1:size(p,1)
    for j = 1:in.numofchrom
        teval(i,j) = evaltree(in.chroms{j}, p(i,:));
    end
end

out = teval*in.coeffs;
```

B.4 Crossover and Mutation

```
function [out1, out2] = gpcrossover(in1, in2)

out1 = in1;
out2 = in2;

len1 = in1.numofchrom;
len2 = in2.numofchrom;

perm1 = randperm(len1);
perm2 = randperm(len2);

nofchange = randi(min(len1, len2));

for i = 1:nofchange
    temp = in1.chroms{perm1(i)};
    out1.chroms{perm1(i)} = in2.chroms{perm2(i)};
    out2.chroms{perm2(i)} = temp;
end
```

```
function out = mutatetree(in, mutationrate, depth)

out = in;
rnd = rand;
if rnd<=mutationrate
    out = maketree(depth);
else
    switch in.type
        case 0 % terminal
            return
        otherwise % operator
            if length(in.operand)==1
                out.operand = mutatetree(in.operand, mutationrate,
depth-1);
            else
                out.operand = {mutatetree(in.operand{1},
mutationrate, depth-1), ...
                               mutatetree(in.operand{2},
mutationrate, depth-1)};
            end
    end
end
```