COMPUTATIONAL INTELLIGENCE TECHNIQUES FOR DATA ANALYSIS

ANG JI HUA, BRIAN

(B.Eng.(Hons.), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2009

Acknowledgements

I would like to express my deepest gratitude to my main supervisor Associate Professor Tan Kay Chen for his continuous motivation and encouragement throughout my Ph.D. candidature. He has also given me lots of valuable advice and guidance along the way. I would also like to highlight and thank my co-supervisor Associate Professor Abdullah Al Mamun for his patience and guidance, in addition to the concern that has been shown to me.

I would not forget the laboratory mates whom I have spent so much time together with, they are Chi Keong, Dasheng, Eu Jin, Chiam, Chun Yew, Han Yang, Chin Hiong and Vui Ann. Not only have they been a bunch of great laboratory mates but also as very great friends. Special thanks to the laboratory officers of the Control and Simulation Laboratory, Sara and Hengwei, for the logistic and technical support.

Many thanks and hugs to my family members for their unremitting support and understanding. Last but not least, I would like to thank all those whom I have unintentionally left out but in one way or another accompanied me through and helped me during my stay at the National University of Singapore.

Publications

Journal Papers

1. J. H. Ang, K. C. Tan and A. A. Mamun, "An Evolutionary Memetic Algorithm for Rule Extraction", *Expert Systems with Applications*, accepted.

2. J. H. Ang, K. C. Tan and A. A. Mamun, "Training Neural Networks for Classification using Growth Probability-Based Evolution", *Neurocomputing*, vol. 71, pp. 3493–3508, 2008.

3. J. H. Ang, S-U. Guan, K. C. Tan and A. A. Mamun, "Interference-Less Neural Network Training", *Neurocomputing*, vol. 71, pp. 3509–3524, 2008.

4. K. C. Tan, Q. Yu and J. H. Ang, "A Coevolutionary Algorithm for Rules Discovery in Data Mining", *International Journal of Systems Science*, vol. 37, no. 12, pp. 835-864, 2006.

5. K. C. Tan, Q. Yu and J. H. Ang, "A Dual-Objective Evolutionary Algorithm for Rules Extraction in Data Mining", *Computational Optimization and Applications*, vol. 34, pp. 273-294, 2006.

6. S-U. Guan and J. H. Ang, "Incremental Training based on Input Space Partitioning and Ordered Attribute Presentation with Backward Elimination", *Journal of Intelligent Systems*, vol. 14, no. 4, pp. 321-351, 2005.

Book Chapters

1. S-U. Guan, J. H. Ang, K. C. Tan and A. A. Mamun, "Incremental Neural Network Training for Medical Diagnosis", *Encyclopedia of Healthcare Information Systems*, Idea Group Inc., N. Wickramasinghe and E. Geisler (Eds.), vol. II, pp. 720-731, 2008.

2. J. H. Ang, C. K. Goh, E. J. Teoh and K. C. Tan, "Designing a Recurrent Neural Network-Based Controller for Gyro-Mirror Line-of-Sight Stabilization System using an Artificial Immune Algorithm", *Advances in Evolutionary Computing for System Design*, Springer, L. C. Jain, V. Palade and D. Srinivasan (Eds.), pp. 189-209, 2007.

Conference Papers

1. J. H. Ang, K. C. Tan and A. A. Mamun, "A Memetic Evolutionary Search Algorithm with Variable Length Chromosome for Rule Extraction", in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, Singapore, pp. 535-540, October 12-15, 2008.

2. J. H. Ang, C. K. Goh, E. J. Teoh and A. A. Mamun, "Multi-Objective Evolutionary Recurrent Neural Networks for System Identification", in *Proceedings of IEEE Congress on Evolutionary Computation*, Singapore, pp.1586-1592, September 25-28, 2007.

Table of Contents

Summary

Due to an increasing emphasis on information technology and the availability of larger storage devices, the amount of data collected in various industries has snowballed to a size that is unmanageable by human analysis. This phenomenon has created a greater reliance on the use of automated systems as a more cost effective technique for data analysis. Therefore, the field of automated data analysis has emerged as an important area of applied research in recent years.

Computational Intelligence (CI) being a branch of Artificial Intelligence (AI) is a relatively new paradigm which has been gaining increasing interest from researchers. CI techniques consist of algorithms like, Neural Networks (NN), Evolutionary Computation (EC), Fuzzy Systems (FS), etc. Currently, CI techniques are only used to complement human decisions or activities; however, there are visions that over time, it would be able to take on a greater role.

The main contribution of this thesis is to illustrate the use of CI techniques for data analysis, focusing particularly on identifying the existing issues and proposing new and effective algorithms. The CI techniques studied in this thesis can be largely classified into two main approaches, namely non-rule-based approach and rule-based approach. The issues and different aspects of the approaches, in terms of implementation, algorithm designs, etc., are actively discussed throughout, giving a comprehensive illustration on the problems identified and the proposed solutions.

The first chapter of this thesis serves as an introductory chapter which includes the motivations behind the proposed work, a comprehensive survey of the current state-

of-the-art methodologies in literature, the necessary technical background information, and the key concepts required to appreciate this thesis.

Chapter 2 and Chapter 3 then discuss the architectural design issues of NN for classification. In particular, Chapter 2 addresses the problem of the lack of segregation of the input feature space during conventional NN training which often causes interference within the network. In Chapter 3, a novel evolutionary approach, which uses a growth probability, is proposed to optimize the weights and architecture of NN.

Chapter 4 and Chapter 5 then illustrate the rule-based algorithms. In Chapter 4, an Evolutionary Memetic Algorithm (EMA) which uses two different local search schemes to complement the global search capability of Evolutionary Algorithms (EA) is proposed for rule extraction to discover knowledge from data sets. Subsequently, in Chapter 5, a Multi-Objective Rule-Based Technique (MORBT) for time series forecasting is proposed.

Last but not least, Chapter 6 concludes on the work presented in this thesis. As several possible areas of exploration within the field are still promising and useful, future directions are also given.

List of Tables

List of Figures

List of Acronyms

AI	Artificial Intelligence
AIRS	Artificial Immune Recognition Systems
AIS	Artificial Immune Systems
ARMA	Autoregressive Moving Average
BP	Backpropagation
CBP	Constructive Backpropagation
CC	Cascade-Correlation
CDA	Co-Discrimination Ability
CI	Computational Intelligence
DA	Discriminant Analysis
DNC	Dynamic Node Creation
DT	Decision Trees
EA	Evolutionary Algorithms
EC	Evolutionary Computation
EMA	Evolutionary Memetic Algorithm
ENN	Evolutionary Neural Networks
EP	Evolutionary Programming
ES	Evolutionary Strategies
FL	Front Limit
FS	Fuzzy Systems
FTSE	Financial Times Stock Exchange
FTSF	Financial Time Series Forecasting
GA	Genetic Algorithms
GP	Genetic Programming
IDA	Individual Discrimination Ability
ILIA	Incremental Learning in terms of Input Attributes

ILNNT	Interference-Less Neural Network Training
KDD	Knowledge Discovery in Databases
KNN	*K*-Nearest Neighbor
LS	Local Search
LSA	Least-Squares Analysis
LTM	Long Term Memory
MGNN	Mutation-based Genetic Neural Network
MLP	Multi-Layer Perceptrons
MO	Multi-Objective
MOAIS	Multi-Objective Artificial Immune Systems
MOEA	Multi-Objective Evolutionary Algorithms
MORBT	Multi-Objective Rule-Based Technique
MPANN	Memetic Pareto Artificial Neural Network
MSSE	Mean Sum of Squared Error
MWFF	Modified Weighted Fitness Function
NASDAQ	National Association of Securities Dealers Automated Quotations
NN	Neural Networks
NN-GP	Neural Networks-Growth Probability
NN-SAGP	Neural Networks-Self-Adaptive Growth Probability
RAN	Resource Allocation Network
RBF	Radial Basis Functions
RMSE	Root Mean Squared Error
S&P 500	Standard & Poor's 500
SOM	Self-Organizing Maps
SP-MORBT	Single Phase-Multi-Objective Rule-Based Technique
SSE	Sum of Squared Error
SVM	Support Vector Machines
TSF	Time Series Forecasting
UCI	University of California, Irvine

Chapter 1

Introduction

The use of automated systems for data analysis is an efficient method which reduces cost and provides prompt analysis. The information derived from automated systems is particularly useful to compliment and expedite human decisions.

Several automated and statistical techniques for data analysis have been studied in the literature, which includes *K*-Nearest Neighbor (KNN) [94], Discriminant Analysis (DA), Decision Trees (DT) [188] and Computational Intelligence (CI) techniques like the Neural Networks (NN), Evolutionary Algorithms (EA) [19][161][162] and Fuzzy Systems (FS) [12][106]. This thesis focus on some of the biologically inspired methodologies of CI techniques and displays the different approaches for data analysis. The proposed algorithms in this thesis are used for classification and time series forecasting. Classification is about making decisions and is evident in our daily life, e.g., to a doctor, the decision is to decide if a patient has an illness and to a financial trader, the decision might be to buy, sell or hold an equity. To solve the problem of classification, an algorithm aims to classify instances of data sets into different output classes. On the other hand, time series forecasting is to predict future values based on past values.

Due to the high predictive accuracy and parallel processing capability of NN, it has been widely used for classification in various domains [1][61][120][135][146][173]. However, the classical method of training NN is to present all input features together without any input space segregation. Chapter 2

shows that training conflicting features together would cause interference and deteriorate network performance. An improved NN architecture with reduced interference in the input space is being proposed.

Another disadvantage of the classical method of training NN is the higher probability of getting trapped in local optima due to the gradient-based search techniques employed for weights update. On the other hand, EA due to their global search capability are less likely to get trapped in local optima. Therefore, EA seems to be an excellent candidate to be hybridized with NN [5][50][78][100][186] to improve NN overall performance. While the back bone of classification process follows the working mechanism of NN, the weights and architecture of the NN are optimized by the EA. Chapter 3 proposes a novel evolutionary approach, which incorporated a growth probability, to evolve the near optimal weights and architecture of NN.

Neural networks are often used as non-rule-based classification systems. Though users are able to read the inputs and know the end results, they are not able to extract any linguistic information from the procedure. They are often seen as a "black box" for data analysis as there is no output of any comprehensible information and this has been considered as one of the major drawbacks. With the development of rule extraction (decompositional or pedagogical) from trained NN [109][140][170][172], this has opened up new perspectives as they have the ability of explaining the classification process giving new insights to the data and provide a better understanding of the problem to improve the quality of decisions made. However, rule extraction from NN is a two step process, the first step is to train the NN and the second step is to extract the rules. When the results are not satisfactory, several implications are often involved, e.g., whether the NN is being trained well or whether the rules extracted are representative of the network.

On the other hand, rule-based algorithms like C4.5 [132], DT, FS and EA [19][161][162] exhibit tremendous advantages over black box methods as they represent solutions in the form of high level linguistic rules. Information extracted from databases is only useful if it can be presented in the form of explicit knowledge, like high level linguistic rules, which clearly interprets the data. EA stand out as a promising search algorithm among these rule-based techniques in various fields due to their easy implementation using chromosome structure representation and its population-based global search optimization capability. The ease of representing the rules using chromosome structure for a given problem provides additional flexibility and adaptability. The genotype representation of EA in terms of chromosome structure encodes a set of parameters of the problem to be optimized which allows flexibility in designing the problem representation. Ideally, the representation should clearly reflect the parameters to be optimized, be easy to implement, comprehend and manipulate in order to explore the different issues of the problem well. In addition, EA are able to perform multiple searches concurrently in a stochastic manner, allowing it to converge promptly towards the global optimum. Hence, this non-mathematical complex optimization method has been widely accepted by various researchers as an alternative to classical methodologies.

Therefore in Chapter 4, an Evolutionary Memetic Algorithm (EMA) is proposed for linguistic rule extraction to discover knowledge from data sets. Two local search schemes are used, of which one is inspired by Artificial Immune Systems (AIS), where the concept of clonal selection principle is used. Following Chapter 4, Chapter 5 proposes a Multi-Objective Rule-Based Technique (MORBT) for Time Series Forecasting (TSF).

Last but not least, conclusions together with possible future work and directions are given in Chapter 6.

The following sections in this chapter will introduce the fundamental concepts and relevant materials required to appreciate the work proposed in this thesis.

1.1 Artificial Neural Networks

The principles of artificial neural networks (also commonly known as neural networks in short) follow the working mechanisms of the human brain which is a part of the central nervous systems. The human brain is highly complex and has the ability to compute very difficult problems. Neural networks through the use of connection weights and hidden units (the term hidden units are used interchangeably with hidden neurons in this thesis) mimic the synapses and neurons of the human brain. Neural networks are able to acquire knowledge through learning. The knowledge learnt is stored in the inter-neuron connection weights [72].

1.1.1 Neural Network Architecture

Several types of NN are presented in the literature, these includes Multi-Layer Perceptrons (MLP), Radial Basis Functions (RBF), Support Vector Machines (SVM), Self-Organizing Maps (SOM), etc [72][84][89][143]. Different types of NN are suitable for different applications. This thesis considers the MLP which represents one of the most widely used and effective NN for classification.

The basic building block of the MLP is the neuron (Figure 1.1). A value, which equals to the weighted sum of all the inputs and a bias, is passed through an activation function to produce the output.

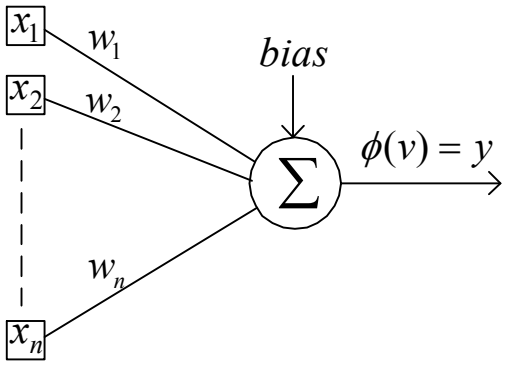


Figure 1.1: A neuron

In Figure 1.1, x_i, $i \in \{1, n\}$, is the *i*th input and *n* is the total number of inputs. w_i is the weight factor corresponding to the *i*th input, v is the weighted sum of all the inputs plus bias, $\phi(.)$ is the activation function and y is the output value. The common types of activation functions used are the linear, threshold and the sigmoid functions, given in Equations 1.1, 1.2 and 1.3, respectively. For the linear function, the output would simply be the weighted sum of inputs plus bias.

$$\phi(v) = v, \quad \forall v = \Re \tag{1.1}$$

$$\phi(v) = \begin{cases} 1, & \text{if } v \geq 0 \\ 0, & \text{if } v < 0 \end{cases} \tag{1.2}$$

$$\phi(v) = \frac{1}{1 + e^{-av}} \tag{1.3}$$

where a is a constant.

The MLP is made up of one or several layers of neurons (Figure 1.2). These layers of neurons are commonly known as hidden layers as the computation of the weights are usually hidden from the users. What the users see are the inputs and the end results only.

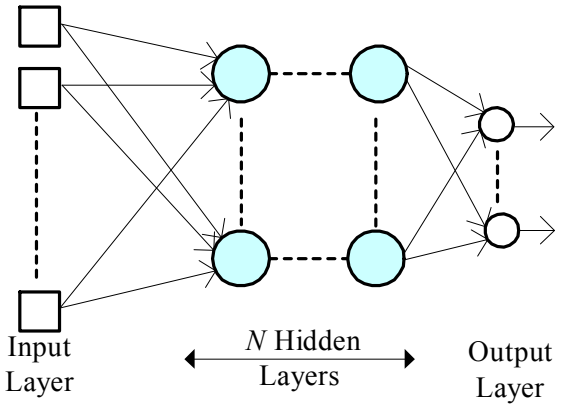


Figure 1.2: Multi-layer perceptrons

1.1.2 Neural Network Training

The inputs are first multiplied by the weight vector and summed with the bias. The weighted sum of the *j*th neuron in hidden layer 1, v_j^1, is given in Equation 1.4.

$$v_j^1 = \sum_{i=1}^{n} x_i w_{ji}^1 + b_j^1 \tag{1.4}$$

This is done for all the neurons in layer 1 and the outputs of the neurons in layer 1 are fed as inputs to the neurons in layer 2. This process is repeated as it propagates through the layers. Eventually, the outputs of the neurons in the last hidden layer are used as inputs to the output units. This process is called the forward pass of the training phase. At the output unit, the resulting error of the network is calculated as

$$e_o = (d_o - y_o) \tag{1.5}$$

where e_o is the error, d_o is the desired output value and y_o is the network output for the *o*th output unit. This error is calculated for all output units and adjustments are made to the weights to minimize the error. There are two methods of updating the weights of the network. One is the batch mode, where updating of the weights is made

after all the input samples are presented. The other method is the sequential mode where updating of the weights is done after every input sample is presented.

This section has provided a brief explanation on the NN training, for a more comprehensive understanding of the NN training process, please refer to [72].

1.1.3 Applications

The uses of MLP [96][98][137][152] can be seen for function approximation, classification, feature selection, etc. Function approximation includes time series forecasting and regression problems.

Depending on the application and the domain, the NN architecture is usually modified to suit the problem. For a time series forecasting problem, the number of input units would usually correspond to the sliding window length, and there would only be one output unit. For a classification problem, the inputs of the NN are the input features and the number of input units would depend on the number of features in the data set. The output units would depend on the number of classes of the problem. If it is a binary class problem, only one output unit is required. The output unit can use the threshold function as given in Equation 1.2 and assign a value to each class. For a multi-class problem, the number of output units can correspond to the number of output classes and a winner-take-all strategy is then used, i.e., the output class is decided on the output unit that has the highest value. These examples are just some of the possible NN representations.

1.2 Evolutionary Algorithms

Evolutionary Algorithms are part of Evolutionary Computation (EC) and it consists of Genetic Algorithms (GA), Genetic Programming (GP), Evolutionary

Strategies (ES) and Evolutionary Programming (EP) [17][37][74][142][174][183][185]. EA follow the natural biological evolution of offspring creation, modification processes and selection procedures to improve the overall fitness of the population over the generations. Operators like the mutation is required to create diversity within the population to escape from local optima. Selection of offspring for next generation is based on the principle of survival of the fittest. Through the use of these simple procedures, EA are able to evolve near optimal solutions for many optimization problems. The flowchart of a typical EA is shown in Figure 1.3.

Evolutionary algorithms are mainly used for optimization problems and more recently, Multi-Objective Evolutionary Algorithms (MOEA) [22][33] are used to optimize several objectives which are often conflicting.

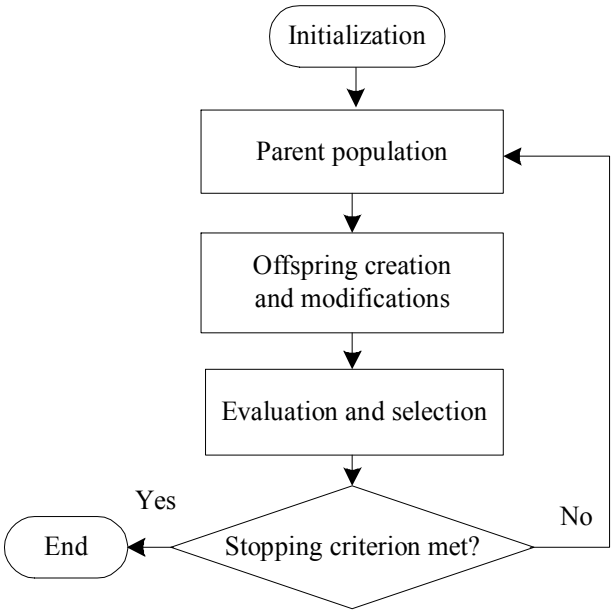


Figure 1.3: Flow chart of a typical evolutionary algorithm

1.3 Rule-Based Knowledge

In order for the users to easily understand the knowledge extracted from data, one method used by EA is to present the extracted information using rules. There are basically two types of rule encoding schemes employed by EA. The encoding of chromosomes to represent a single rule is known as the Michigan encoding. An example of a rule is given as [113][159][162],

$$\textit{If } \boldsymbol{A_1} \text{ and } \boldsymbol{A_2} \text{ and } \ldots \boldsymbol{A_n} \textit{ then } \boldsymbol{C}.$$

where $\boldsymbol{A}_i$, $\forall\, i \in \{1, 2, \ldots, n\}$, is the antecedent of the rule for the ith input and n is the total number of inputs. $\boldsymbol{C}$ is the consequence or the prediction of the rule. The antecedents are seen as the independent variables while the consequences are the dependent variables. This type of rule is interpreted as samples having inputs that match the antecedents would have the following consequence. In a population of Michigan rules, rules are seen as autonomous entities, where each rule classifies the samples independently without being affected by other rules [162]. The insertion or deletion of rules do not influence other rules within the population but would only affect the overall performance of the population. Individual rule within the population is only able to predict a particular class. For a multi-class problem, the coverage of all the output classes would be a collective effort of all the rules.

Another chromosome encoding method (Pittsburgh encoding) to represent the discovered knowledge is in terms of rule sets. A rule set is made up of a variety of different rules and is represented as follows:

If $\boldsymbol{A_{11}}$ and $\boldsymbol{A_{12}}$ and …. $\boldsymbol{A_{1n}}$ *then* $\boldsymbol{C_1}$

else if $\boldsymbol{A_{21}}$ and $\boldsymbol{A_{22}}$ and …. $\boldsymbol{A_{2n}}$ *then* $\boldsymbol{C_2}$

……..

else $\boldsymbol{C}_{general}$

where $\boldsymbol{A}_{ji}$, $\forall\ j \in \{1, 2,\ldots, m\}$, is the antecedent part of the jth rule and m is the maximum number of rules in the rule set. $\boldsymbol{C}_j$ is the consequence of the jth rule. A rule set is different from a single rule in several aspects. Firstly, a rule set is made up of several individual rules, hence it is longer and more complex than a single rule. Secondly, each rule set is able to predict several classes of a problem. Thirdly, the ordering of rules in a rule set is important as it affects the overall performance of a rule set.

When an instance is presented, the first rule of the rule set would be used to classify this instance. If the first rule is not able to classify the instance, i.e., the rule's antecedents do not fit the instance, subsequent rules would be used in the specified order until the instance is classified. If there are no rules that are able to classify the instance, a general class is assigned. When a new instance is being classified by a rule on top, rules at the bottom of the rule set would not have the chance to classify it, thus it is important that rules at the top of the rule sets are good rules.

Both the Michigan and Pittsburgh encoding have their own advantages and disadvantages. The Michigan approach presents a clear and simple rule encoding technique which targets covering a specific region of the search space. Since the search is confined to finding good solutions for a particular class, the search space is much smaller as compared to the Pittsburgh approach. Typically, this results in faster convergence of the EA and higher training fitness for each rule. On the other hand, Pittsburgh encoding results in a more complex interpretation as the rule set is

supposed to cover several possible outcomes. The search space of the Pittsburgh approach is not only enlarged due to finding the overall solution to the problem but also to discover an optimal combination and sequence of the individual rules. However, the consideration of the rule components within the rule set inherently allows Pittsburgh encoding to consider rule interaction, which is its main advantage. This rule interaction is absent in Michigan encoding scheme as individual rule actions and feedback are independent of the rest of the rules in the population [53][79][122].

1.4 Types of Data Analysis

Two types of data are used in this thesis. The first type is the classification data while the second type is the time series data. The main aim of classification is to predict the output classes based on the given inputs. Algorithms for classification are presented in Chapter 2, Chapter 3 and Chapter 4. On the other hand, time series forecasting aims to predict the future values based on previous observations. Chapter 5 presents the algorithm for time series forecasting.

1.4.1 Classification

This section presents the overview of classification and the data sets used in this thesis.

1.4.1.1 Overview

Classification is part of the Knowledge Discovery in Databases (KDD) process [115][133]. The whole process of KDD starts from the collection of raw data to extraction of the knowledge. Raw data is collected by measurements and these data are then preprocessed before introducing into the data mining algorithms. In some

cases, as not all the data collected are useful for the specific purpose, data reduction in terms of feature selection is done. Projection to lower dimension can also be done to reduce the input complexity. These data are passed through data mining algorithms for data analysis. Different types of analysis are done depending on the nature of the data. The analysis is then evaluated and the given knowledge is able to assist expert decisions. The flowchart is given Figure 1.4.

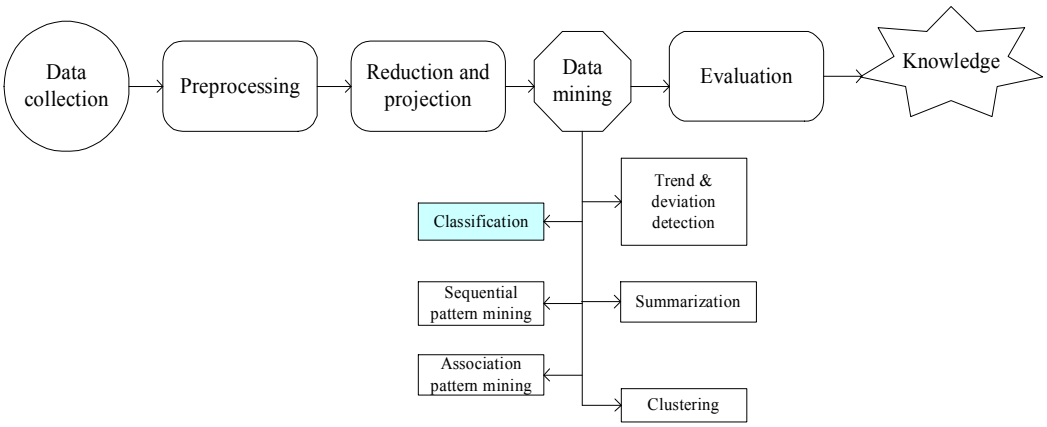


Figure 1.4: Knowledge discovery process

1.4.1.2 Classification Data Sets

These data sets are taken from the University of California, Irvine (UCI) machine learning repository [18] benchmark collection. The data are collected from real-world problems. Some of the data sets are pre-processed by PROBEN1 benchmark collection [130]. The details of the data sets are given below:

Cancer Data Set: The objective of the cancer problem is to diagnose breast cancer in patients by classifying a tumor as benign or malignant. The "Breast Cancer Wisconsin" problem data set was originally collected in the University of Wisconsin Hospitals, Madison, by Dr. William H. Wolberg [108]. 458(65.5%) of the samples in the data set are benign while 241(34.5%) of the samples are malignant. There are nine

attributes as given in Table 1.1.

Table 1.1: Attribute descriptions – cancer

No.	Attribute descriptions
1	Clump thickness
2	Uniformity of cell size
3	Uniformity of cell shape
4	Marginal adhesion
5	Single epithelial cell size
6	Bare nuclei
7	Bland chromatin
8	Normal nucleoli
9	Mitoses

Card Data Set: This data set in PROBEN1 benchmark collection contains data collected for credit card applications. The problem is to decide whether approval should be given to a credit card application. The "crx" data of the credit screening problem in the UCI machine learning repository was used to create this data set. Description for each attribute is not disclosed, for confidentiality reasons, in the original data set. There are a total of 15 attributes, 2 outputs and 690 instances. The class distribution is 44.5% (307 out of 690 instances) of applications are granted approval and 55.5% (383 out of 690 instances) denied.

Diabetes Data Set: The diabetes problem is to diagnose a Pima Indian individual based on personal data and medical examination for diabetes. For this data set, 500 (65.1%) samples do not have diabetes. There are eight attributes and two output classes. The descriptions of the attributes are shown in Table 1.2.

Table 1.2: Attribute descriptions – diabetes

No.	Attribute descriptions
1	Number of times pregnant
2	Plasma glucose concentration
3	Diastolic blood pressure
4	Triceps skin fold thickness
5	2-Hour serum insulin
6	Body mass index
7	Diabetes pedigree function
8	Age

Glass Data Set: Glass classification is extremely useful in criminological investigation, as the glass left behind at the crime scene could be used as criminal evidence if they are correctly classified. The problem is to classify glasses into one of the six different glass types based on chemical properties content. The input attributes for this problem are shown in Table 1.3.

Table 1.3: Attribute descriptions – glass

No.	Attribute descriptions
1	Refractive index
2	Na: Sodium
3	Mg: Magnesium
4	Al: Aluminum
5	Si: Silicon
6	K: Potassium
7	Ca: Calcium
8	Ba: Barium
9	Fe: Iron

Heart Data Set: The heart problem data was collected from the Cleveland Clinic Foundation by principle investigator Andras Janosi, M. D., Hungarian Institute of Cardiology, Budapest. There are 13 attributes in this problem and the goal is to determine if heart disease is present in a patient. 54.1% (164) of the examples do not have heart disease, while 45.9% (139) of the examples show presence of heart disease. The descriptions of the attributes are shown in Table 1.4.

Table 1.4: Attribute descriptions – heart

No.	Attribute descriptions
1	Age
2	Sex
3	Chest pain type
4	Resting blood pressure
5	Serum cholesterol
6	Fasting blood sugar
7	Electrocardiographic results
8	Maximum heart rate
9	Exercised induced angina
10	ST depression
11	Peak ST segment slope
12	No. of major vessels
13	Thal

Iris Data Set: The iris data set is a multi-class botanical problem and has been one of the most widely used data set in the pattern recognition literature [44]. Among the three classes, one class is linearly separable from the other two classes while these two classes are non-linearly separable. The three different types of iris plant to be identified are the iris setosa, versicolour and virginica.

Table 1.5: Attribute descriptions – iris

No.	Attribute descriptions
1	Sepal length
2	Sepal width
3	Petal length
4	Petal width

Soybean Data Set: This data set aims to identify 19 different diseases of soybean based on bean and plant physical descriptions and information regarding plant's life history. There are 35 input features, 19 output features and 683 examples. This data set has the largest number of classes in PROBEN1 benchmark collection [130]. As there are too many features, the details are not listed here. For details of the data set, please refer to [130].

1.4.2 Time Series Forecasting

This section presents the overview of time series forecasting and the type of time series used in this thesis.

1.4.2.1 Overview

Patterns of time series data are often categorized as trend or seasonal. Trend patterns do not have patterns repeating within the observed time period, while seasonal patterns would have similar patterns for some period within the observations. Typically, time series data are often erroneous, noisy and filled with outliers. Simple techniques, like the moving average with a predefined averaging window size [21], are used for filtering.

A time series is represented by $X = [x_1, x_2, ..., x_n]$, where n is the number of observations and $[x_i, x_{i+1}]$ is equally spaced observations in time, $\forall i = 1, ..., n-1$. The basic working mechanism of TSF lies in the assumption that a consecutive sequence of observations in time is representative of the successive observations. In this case, the primary objective of TSF techniques is to first derive the relationship of the current observations, then based on the identified relationship, predict the future output values. Mathematically, using $[x_i, x_{i+w-1}]$ observations to predict $x_{i+w-1+q}$, where w is the sliding window length (i.e., number of time steps used for prediction) and q is the number of steps ahead to predict.

1.4.2.2 Financial Time Series

Stock market indices, being one of the main indicators of stock valuation and consumers' sentiments of companies, could be largely fluctuating as it is highly responsive to the current economic outlook. Amidst the large movements and often

unexpected changes, there lie substantial opportunities for investors to gain from the market. Several studies have started in this field, this includes trading strategies, financial time series forecasting and portfolio optimization [14][32][43][86][126][141][151]. The use of mathematical models and intelligent systems would be able to provide prompt and better understanding of the market trend [55].

In Financial Time Series Forecasting (FTSF), technical analysts hope to derive some relationship among past and current market data and present it for future uses and gains. The ability to predict the future prices of equities and market indices is pertinent for fund managers to make sound decisions. However, numerous factors could affect the market indices, with large capital flows exchanged between different institutions, banks and retailers in day to day trading activities. With these activities, huge amount of data are accumulated to be analyzed. The area of financial index prediction has always been a sought after research area and in recent years it is gaining significant attention from researchers in the field of TSF.

The data sets used in this thesis are the main indexes in the London and United States stock exchange markets. The algorithm in Chapter 5 is applied on the Financial Times Stock Exchange (FTSE) index, Standard & Poor's 500 (S&P 500) index and the National Association of Securities Dealers Automated Quotations (NASDAQ) index. The FTSE represents the index for the most highly capitalized companies on the London stock exchange, while the S&P 500 consists of 500 large market capitalization corporations in the United States market.

1.5 Contributions

The main contribution of this thesis is to illustrate the use of CI techniques for data analysis, focusing particularly on identifying the current issues and proposing new and effective algorithms. The techniques studied can be broadly grouped into non-rule-based and rule-based approaches.

For non-rule-based approach, the architectural design issues of NN are discussed. In Chapter 2, the lack of partitioning of the input space in conventional NN training is investigated. An improved NN architecture with reduced interference in the input space is then proposed. In Chapter 3, a novel EA, which uses a growth probability, is proposed to optimize the architecture and weights of NN.

For rule-based approach, Chapter 4 proposed an Evolutionary Memetic Algorithm (EMA) for rule extraction to discover knowledge from data sets. In EMA, two different local search schemes are used to complement the global search capability of EA. In Chapter 5, a multi-objective optimization algorithm, incorporated within a dual phase framework, is proposed to evolve rules for TSF.

In general, the proposed algorithms have shown to be effective for data sets that spread over a variety of fields. The algorithms produced results that are generally good and comparable to those in existing literature.

The contributions and motivation for each proposed algorithm will be given in further details in the respective chapters.

Chapter 2

Interference-Less Neural Network Training

In classical methods of NN training, all the input attributes are connected to the same hidden neurons and these attributes are introduced together to the network for training concurrently. However, these input attributes have different levels of classification abilities with some having higher discrimination factor. In addition, different attributes have different classification criteria and attributes will interfere in the decision making of others if all attributes are trained in the same batch concurrently. Interference among attributes leading to poor accuracy might arise because attributes under training are affected by the decisions of other attributes that are inconsistent with theirs. Many real-world data consist of conflicting information [85], causing the network to take a long time to decide its direction and thus affecting the accuracy of output results. It is important to ensure learning by some input attributes is not undone by other attributes [181].

The performance of NN could be influenced by several factors like network architecture, training algorithm, etc. In particular, the input space architecture is of great importance [62][63][64][65][66][76][128]. For neuro-fuzzy networks approaches [90], grid partitioning is applied to the input data of the data set in order to generate an initial fuzzy inference system. Recursive partitioning in input space is applied to overcome the limitations of conventional neuro-fuzzy systems [175]. [148] partitions the input space into different regions and applies differential weighting for

different regions so that they have different agents that are specialized in local regions. These works however do not investigate the interference that exists in the networks and the interference among the attributes constitutes a major setback to the classification ability of the NN.

Weaver et al. [181] mitigates the interference (learning in one part of the network causing unlearning in the other part) effect by reduction of a bi-objective cost function that combines the approximation error and a term that measures interference to adjust the weights of an arbitrary, non-linearly parameterized network. [88] investigates the interference (learning due to new samples causing unlearning of the old samples) caused by training NN when data are presented incrementally, i.e., data samples are shown sequentially. A Long Term Memory (LTM) is incorporated into Resource Allocation Network (RAN), i.e., the network will train all the new data and part of the old data. Though these works investigated the interference, they do not deal directly with the input space as previously discussed as an important factor affecting network performance. They do not determine the interference between the input attributes and devise an attribute partitioning algorithm to avoid interference.

In this chapter, Interference-Less Neural Network Training (ILNNT) algorithm which determines the interfering relationship between input attributes, and partitions them accordingly to this relationship, is studied in detail. Two attributes are first analyzed for interference. Using the partitioning algorithm, mutually benefiting attributes are grouped together to maximize the information contained in them while interfering attributes are separated and trained under different sub-networks. It is usually difficult to decide which batch a particular attribute should belong to, as an attribute can belong to several batches. The algorithm proposed in this chapter tries to make grouping as comprehensive as possible. The architecture of ILNNT is built

upon an incremental NN as this network construction technique is suitable for the application of the interference-less algorithm. There are several incremental NN available in the literature [61][66][147] and the architecture of ILNNT is built upon ILIA (Incremental Learning in terms of Input Attributes) [61].

This chapter is divided into 6 sections. The next section states the Constructive Backpropagation (CBP) Learning Algorithm and Section 2.2 describes the incremental NN used for ILNNT. In Section 2.3, the details of ILNNT are presented. Section 2.4 states the experimental setup and data sets used. The experimental results of different data sets are given and analyzed in Section 2.5. Finally, the conclusions are given in Section 2.6.

2.1 Constructive Backpropagation Learning Algorithm

The architecture of ILNNT is built upon ILIA [61] and the incremental algorithm adopted the constructive backpropagation learning algorithm [93] to train the weights and to determine the number of hidden neurons needed for each of its sub-network. The CBP learning algorithm can be briefly described in the following steps:

Step 1: Apply direct connections from the input units to the output units, and initialize this network with bias weights. Training of the weights is by minimizing the sum of squared error (Equation 2.1). Values of the weights at the end of training are then fixed. No hidden units (hidden neurons) are installed in this initial network.

$$Error = \sum_{i=1}^{m}\sum_{j=1}^{o}(a_{ij} - d_{ij})^2 \tag{2.1}$$

where m = total number of training examples, o = number of outputs for the problem, a_{ij} = actual network jth output unit value for the ith training example, and d_{ij} = desired network jth output unit value for the ith training example.

Step 2: Add a new hidden unit to the network (nth unit, $n > 0$). Connect the input units and output units to this new hidden unit. Training of the weights connected to the new hidden unit then uses the modified sum of squared error.

$$ModifiedError_n = \sum_{i=1}^{m}\sum_{j=1}^{o}\left(d_{ij} - \sum_{k=0}^{n-1}u_{kj}h_{ki} - u_{nj}h_{ni}\right)^2 \tag{2.2}$$

where u_{kj} = connection from the kth hidden unit to the jth output unit, h_{ki} = output of the kth hidden unit for the ith training example ($k = 0$ represents step 1), u_{nj} = connection from the nth hidden unit to the jth output unit, and h_{ni} = output of the nth hidden unit for the ith training example.

Step 3: Fix the weights obtained in step 2.

Step 4: Evaluate the performance of the network. If the performance of the network is acceptable, stop adding more hidden units, else repeat step 2.

2.2 Incremental Neural Networks

Incremental Learning in terms of Input Attributes (ILIA) [61] is a type of constructive NN that is concern with new incoming attributes to the existing pool of attributes. When additional input attributes are to be considered, ILIA expand its input

dimension to accommodate these new attributes while fixing its existing architecture. Learning is done only for the new sub-network formed for the new attributes. ILIA used the CBP learning algorithm and applied the method of *early stopping* using a validation data set to prevent overfitting [130]. Details of ILIA can be found in [61] while a brief description is given in the following sub-sections.

2.2.1 Incremental Learning in terms of Input Attributes 1

Existing available attributes are first presented to the NN as shown in Figure 2.1. This network consists of direct connections from all the input units to all the output units. All the input units are also connected to the same hidden units and eventually the hidden units are connected to all the output units. The CBP learning algorithm [93] is used to train the weights and to determine the number of hidden neurons needed. The weights are fixed when the network is fully trained.

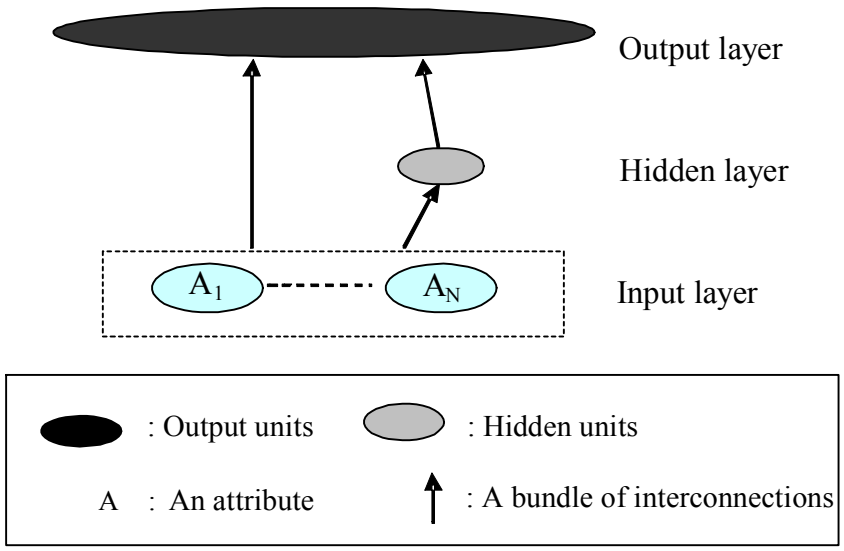


Figure 2.1: Initial network

When there are new attributes to be considered, increase the input dimension to accommodate the new attributes. This is done by adding a new sub-network (in terms

of new direct connections and new hidden units) to the existing network. Figure 2.2 shows the new NN architecture. There are direct connections from all the new input units to all the output units and all the new hidden units. The new hidden units are also connected to all the output units. There are no connections between the new input units and the hidden units of the initial network. The NN obtained here is a single hidden layer feedforward NN. The process of training the weights and installing new hidden units is only done for the new sub-network.

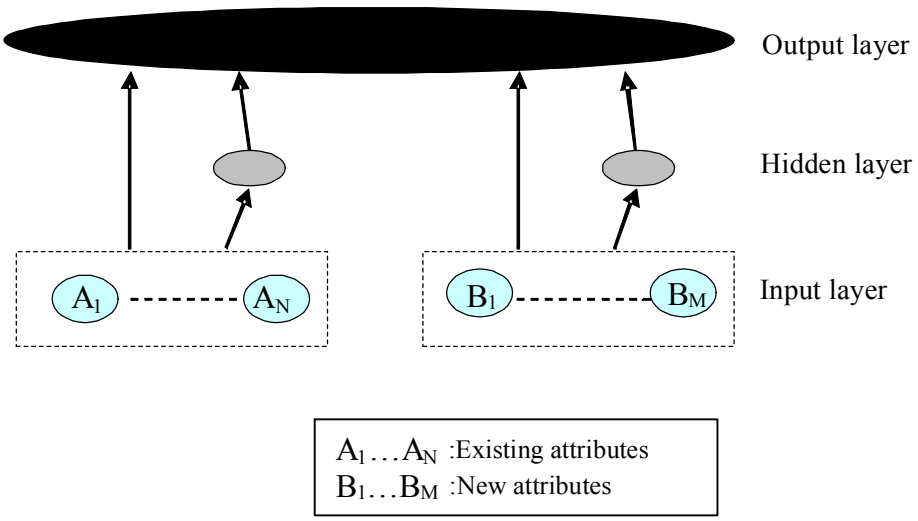


Figure 2.2: Architecture of ILIA1

2.2.2 Incremental Learning in terms of Input Attributes 2

ILIA2 (Figure 2.3) extends ILIA1 by adding a new output layer to the network obtained in ILIA1. The number of new output units added is equal to the number of output units in ILIA1. Thus, the old output layer in ILIA1 is effectively collapsed and becomes a new hidden layer. New connections are created from all input units and new hidden units (previous output units) to the new output units. Eventually, all the newly added connection weights are adjusted. The motivation for ILIA2 is to obtain more information than ILIA1 as it collapses the original output layer, therefore it has

the potential to grasp higher-order information and to 'update' or 'improve' the existing network via the added connections between the new output units and the collapsed output units.

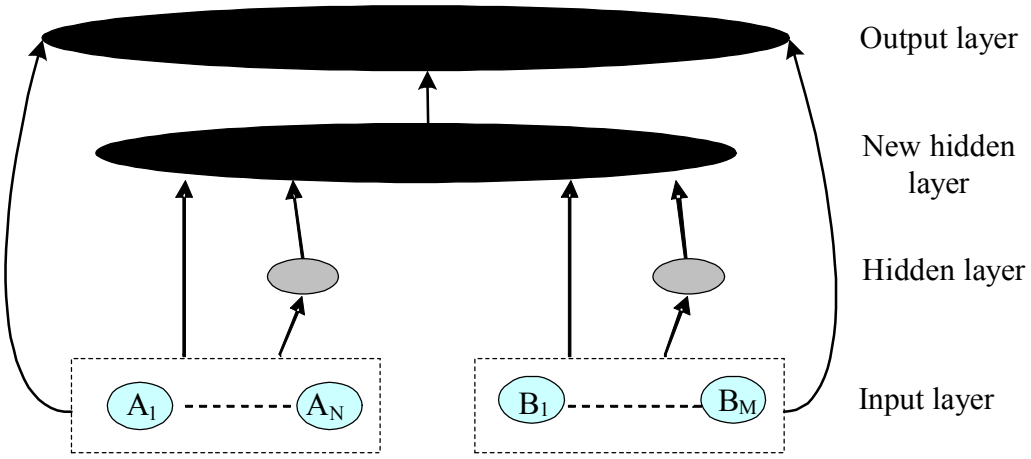


Figure 2.3: Architecture of ILIA2

2.3 Details of Interference-Less Neural Network Training

The ILNNT consists of three main steps; the interference table formulation, interference-less grouping using the partitioning algorithm and training the network in batches. The details of each step are given in the following sub-sections.

2.3.1 Interference Table Formulation

The interference table consists of two main components; the Individual Discrimination Ability (IDA) of an attribute [67] (error of that particular attribute when other attributes are absent from the NN training), and Co-Discrimination Ability (CDA) of any two attributes (error of those two attributes when other attributes are absent from the NN training).

2.3.1.1 Individual Discrimination Ability Evaluation

Figure 2.4 shows the network used to evaluate IDA [67]. The learning algorithm used for this network is the CBP learning algorithm. This NN only has one attribute in the input space. The lower the error attained by the NN, the higher is the discrimination ability of the attribute used by the NN.

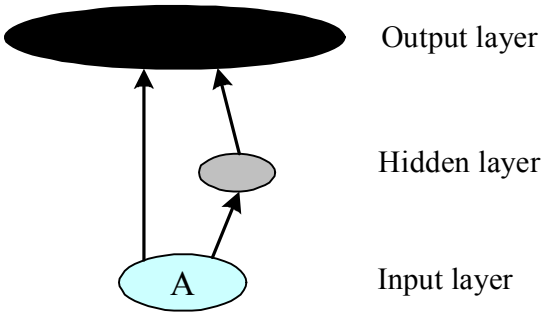


Figure 2.4: Network used to evaluate individual discrimination ability

2.3.1.2 Co-Discrimination Ability Evaluation

Figure 2.5 shows the network used to evaluate the CDA of any two attributes, i.e., error resulting from training the two attributes in the same batch. The input layer will only consist of those two attributes to be tested. All inputs will have direct connections to the output units and to the same hidden units.

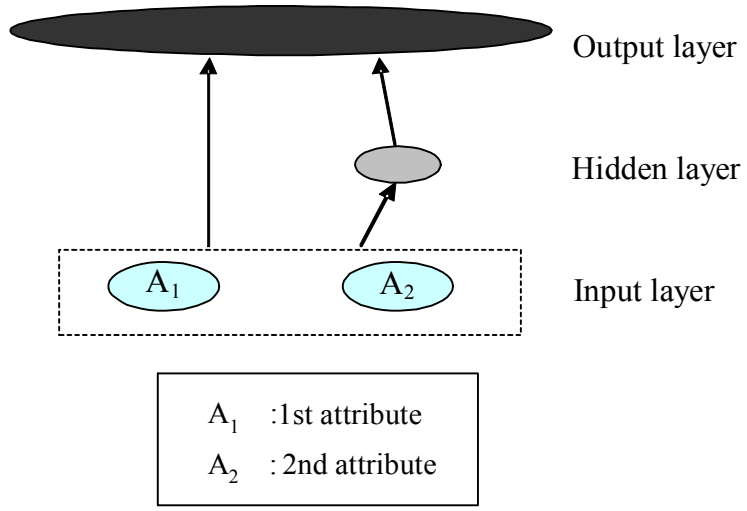


Figure 2.5: Network used to evaluate co-discrimination ability

2.3.1.3 Interference Table

Table 2.1 shows a generic interference table. I_i or I_j represents the IDA and E_{ij} represents the error arising from training attribute *i* and attribute *j* together. If ($E_{ij} > I_i$ or $E_{ij} > I_j$), then attribute *i* and attribute *j* are interfering, else if ($E_{ij} \leq I_i$ and $E_{ij} \leq I_j$), they are non-interfering.

Table 2.1: Interference table

Attribute	**1**	**2**	**3**	**4**	**5**	**6**	……..	**N**
1	I_1							
2	E_{12}	I_2						
3	E_{13}	E_{23}	I_3					
4	E_{14}	E_{24}	E_{34}	I_4				
5	E_{15}	E_{25}	E_{35}	E_{45}	I_5			
6	E_{16}	E_{26}	E_{36}	E_{46}	E_{56}	I_6		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	……..	
N	E_{1N}	E_{2N}	E_{3N}	E_{4N}	E_{5N}	E_{6N}	……..	I_N

To illustrate the concept, Table 2.2 shows the interference table of the diabetes problem. There is interference among some of the attributes. Attribute 4 has significant interference, interfering with half of the other attributes, e.g., attribute 6 IDA is 21.68%, but when trained with attribute 4, it increases to 21.72%. Also from the table, information regarding attributes that are beneficial to other attributes can be determined. Attribute 1 does not interfere with any other attributes, e.g., when trained with attribute 6, error decreases to 20.50% which is lower compared to their IDA.

Table 2.2: Interference table - diabetes

% error	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	**21.74**							
2	16.25	**17.08**						
3	21.73	16.90	**23.27**					
4	21.64	17.14	23.09	**23.14**				
5	21.12	17.01	22.87	22.95	**22.92**			
6	20.50	16.84	21.51	21.72	21.49	**21.68**		
7	21.18	17.09	22.67	22.57	22.49	21.27	**22.73**	
8	19.61	15.04	19.85	20.05	20.11	19.37	20.20	**20.02**

Table 2.3: Summary of the interference table - diabetes

Attribute	Does not interfere with	Interfere with
1	2 3 4 5 6 7 8	-
2	1 3 5 6 8	4 7
3	1 2 4 5 6 7 8	-
4	1 3 7	2 5 6 8
5	1 2 3 6 7	4 8
6	1 2 3 5 7 8	4
7	1 3 4 5 6	2 8
8	1 2 3 6	4 5 7

Notes: The column 'Does not interfere with' contains attributes that have no interference with the attribute in the 'Attribute' column.
The column 'Interfere with' contains attributes that interfere with the attribute in the 'Attribute' column.

2.3.2 Interference-Less Partitioning Algorithm Formulation

Following the determination of interference among the attributes, the interference-less partitioning algorithm is able to draw on this information and make groupings that reduce interference among the attributes while maximizing attributes' co-benefits. In this case, not only the interference among attributes is reduced but the performance of individual attribute can also be boosted by allowing other attributes that will assist it to be trained together. The algorithm is recursive in nature, trying to group as many mutually beneficial attributes together.

2.3.2.1 Partitioning Algorithm

(A) Choose a starting attribute. Set starting attribute as reference attribute, split attributes into 2 batches; interfering and non-interfering. The interfering batch has all the attributes that interfere with the reference attribute while the non-interfering batch has all the attributes that do not interfere. To have a more comprehensive partitioning, every attribute should be used once as the starting attribute after each round. With different starting attribute, different possible partitioning can be achieved. From the different possible partitioning, the network that performs best in the training phase could be applied on unseen data.

(B) Use next attribute in sequence as the reference attribute. Distinguish the batch the reference attribute belongs to. Identify if there are any interfering attributes within the batch. If there are no interfering attributes, terminate step B, else identify the number of interfering attributes with the reference attribute and group them as follows:

- If the number of interfering attributes is greater than the number of non-interfering attributes, take out the reference attribute.

- If the number of interfering attributes is less than or equal to the number of non-interfering attributes, take out the interfering attributes to form a new batch.

Apart from the sequence mentioned, there are other possible sequences to choose the next attribute as the reference attribute, e.g., the attributes could be arranged in order of number of attributes that they interfere with, and the partitioning sequence could be tried for this ordering.

(C) Repeat step B until the reference attribute selection is back to the starting attribute.

(D) Retain the largest batch and group all attributes in the smaller batches together. Nothing is to be done for the largest batch and a new batch is formed from the combination of the smaller batches.

(E) Perform step A to step C to the newly formed batch.

(F) Repeat steps D and E until no new partitioning is achieved.

The flow chart for the algorithm is given in Figure 2.6.

Elimination of interference is done by separating attributes that do not give good accuracy when being grouped together. They produce worse results when compared to individual attributes that are trained alone. The cooperation among attributes is enhanced by grouping those attributes that are mutually beneficial. The algorithm does not separate those attributes whose co-relation is favorable for better classification. When an attribute is beneficial to other attributes in the batch, they are retained. Attributes are only taken out from a batch if there are more interfering attributes than non-interfering attributes. Since the algorithm starts with all attributes in a batch, cooperation is maintained when mutually beneficial attributes are allowed to remain within a batch. In addition, instead of ending the algorithm after one round

of partitioning, the smaller batches are grouped together and the partitioning algorithm is applied again on the newly grouped attributes. This has the effect of forming larger groups for mutually beneficial attributes.

The interference table computation takes into consideration the interference relationship of two attributes. The interference relationship between more attributes taken together could be similarly achieved, but at the expense of higher computational cost. Interference computation could be done intensively for all possible combination of attributes, which in this case would be extremely large and eventually would not need any partitioning algorithm since the grouping is so comprehensive that all possibilities are tested. The algorithm presented in this chapter provides a trade-off between the computation complexity and accuracy, hence termed interference-less rather than interference-free.

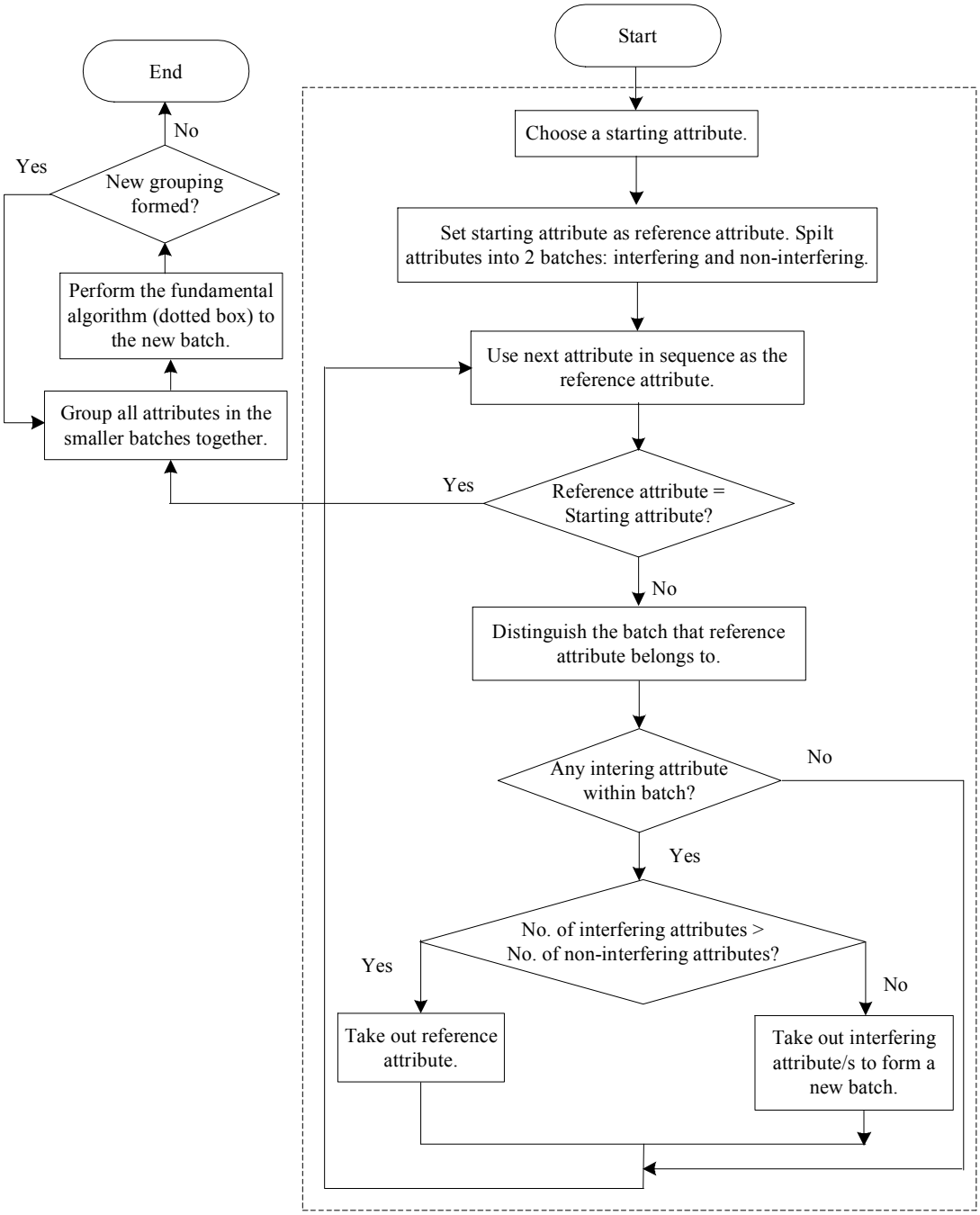


Figure 2.6: Flowchart of interference-less partitioning algorithm

2.3.2.2 An Example – Diabetes Data Set

The diabetes data set contains 8 attributes. Each trial represents the partitioning result when an attribute is used. If there are *n* attributes, a complete fundamental round will consist of *n* trials. Using Table 2.2 or 2.3, the partitioning algorithm proceeds as follows:

Trial 1: As an example, choose attribute 5 as the starting attribute. Using attribute 5 as the reference attribute, split attributes into 2 batches, interfering and non-interfering. The non-interfering batch in this case is (1 2 3 5 6 7) and the interfering batch is (4 8).

Trial 2: Use the next attribute in sequence, i.e., attribute 6 as the reference attribute now. Distinguish the batch that attribute 6 belongs to. Attribute 6 belongs to the batch (1 2 3 5 6 7). Since there are no interfering attributes within the batch, nothing needs to be done. Though attribute 6 interferes with attribute 4, attribute 4 belongs to another batch, thus it is not a concern here. At the end of trial 2, the partitioning remains at (1 2 3 5 6 7) (4 8).

Trial 3: Use attribute 7 as the reference attribute. Attribute 7 belongs to the batch (1 2 3 5 6 7). Attribute 2 is identified as an interfering attribute within the same batch as attribute 7. Within the batch (1 2 3 5 6 7), the number of interfering attributes with attribute 7 is one, whereas the number of non-interfering attributes with attribute 7 is four. The number of interfering attributes is less than the number of non-interfering attributes. Attribute 2 is taken out from the batch. The new grouping is (1 3 5 6 7) (2) (4 8).

Trial 4: Use attribute 8 as the reference attribute. Note that attribute 8 belongs to the batch (4 8). Attribute 4 interferes with attribute 8 and the number of interfering attributes is greater than number of non-interfering attributes.

Attribute 8 is taken out from the batch. The new grouping is (1 3 5 6 7) (2) (4) (8).

Trial 5: Attribute 1 is considered as the next attribute in sequence. Since attribute 1 does not interfere with any attributes within its batch, nothing is to be done.

Trial 6: Attribute 2 is used as the reference attribute. Attribute 2 is a standalone batch by itself, i.e., does not interfere with any attributes within its batch, nothing needs to be done. Grouping remains at (1 3 5 6 7) (2) (4) (8).

Trial 7: Attribute 3 is the reference attribute and since it does not interfere with any attributes within its batch, nothing needs to be done.

Trial 8: Attribute 4 is used as the reference attribute. Attribute 4 is a standalone batch by itself, i.e., does not interfere with any attributes within its batch, nothing needs to be done. Grouping remains at (1 3 5 6 7) (2) (4) (8).

Since attribute 5 is the starting attribute, one fundamental partitioning round is over. Use different starting attribute for different rounds to get different possible input space partitioning.

A sub-round starts after one fundamental partitioning round ends. Group all the smaller batches together; batch (2) (4) (8) will effectively become (2 4 8). The aforementioned partitioning is applied, and the resulting groups will be (2 8) (4). Since no new partitioning can be derived from here on. The final grouping is (1 3 5 6 7) (2 8) (4). If the algorithm had stopped at only one round, mutually benefiting attributes 2 and 8 would not be grouped together. However, due to the recursive nature of the algorithm, this is made possible.

2.3.3 Architecture of Interference-Less Neural Network Training

The incremental learning architecture of ILIA [61] fits the requirement of ILNNT by providing separated training for different batches of attributes. ILNNT aims to have separated training for different batches categorized by interference. The incorporation of ILIA into ILNNT is described below:

Step 1: Begin training for the first batch of attributes, i.e., this forms the first sub-network. The chosen first batch of attributes depends on user's batch ordering. The ordering used in this chapter is the numeric ordering where the batch that contains the attribute with the smallest index would be trained first. This numerical ordering is used for convenience. Other ordering could also be tried and tested, e.g., ordering which is based on larger batches in-front of the queue. This would mean increasing the convergence rate in the early training period and smaller batches would be used for fine-tuning in the late training stage. When the sub-network is fully trained, all the weights are fixed.

Step 2: Introduce the next batch of attributes (based on ordering in step 1) by forming a new sub-network. Training of the weights is only done for this new sub-network. Once again, when the sub-network is fully trained, all the weights are fixed.

Step 3: Repeat step 2 until there are no more batches of attributes not used to train the NN.

The final output of the NN is determined using the winner-take-all strategy, i.e., the decided class goes to the output neuron with the highest output value. The overall architecture of ILNNT1 is shown in Figure 2.7.

Figure 2.8 illustrates an example of ILNNT1 using the resulting partition (1 3 5 6 7) (2 8) (4) of the diabetes data set. The algorithm identifies three batches, which means there will be three sub-networks. Batch 1 consisting of 5 attributes will be train first. After training, the weights are fixed. It is then followed by training batch two (2 attributes) and finally, the last batch.

ILNNT2 extends ILNNT1 by collapsing the output units and adding a new output layer. Thus, the previous output units have effectively become the hidden units in ILNNT2. It is an attempt to improve ILNNT1 for better classification accuracy.

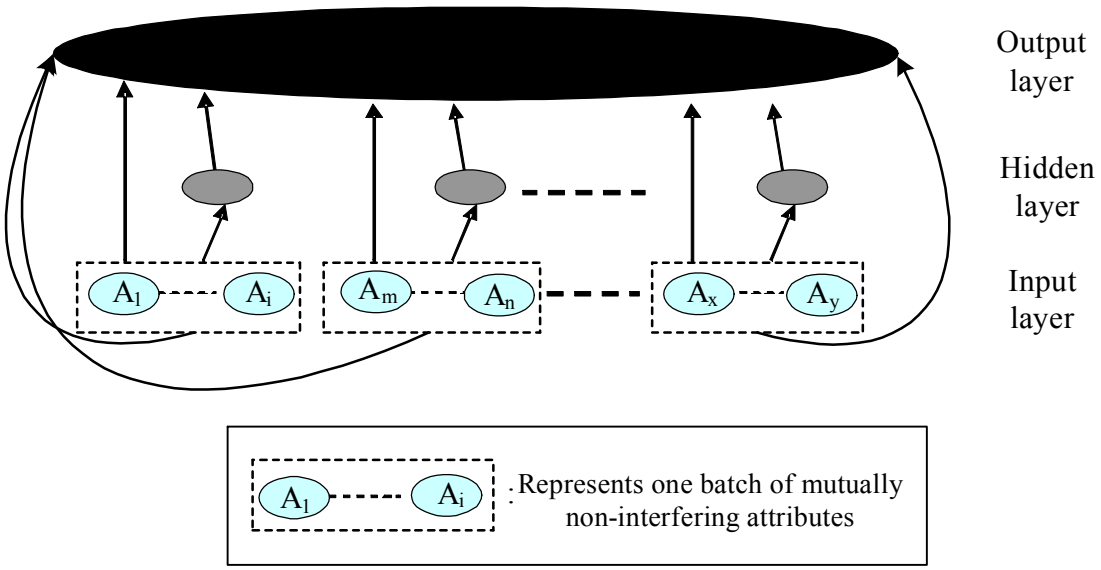


Figure 2.7: ILNNT1 (ILIA1) architecture

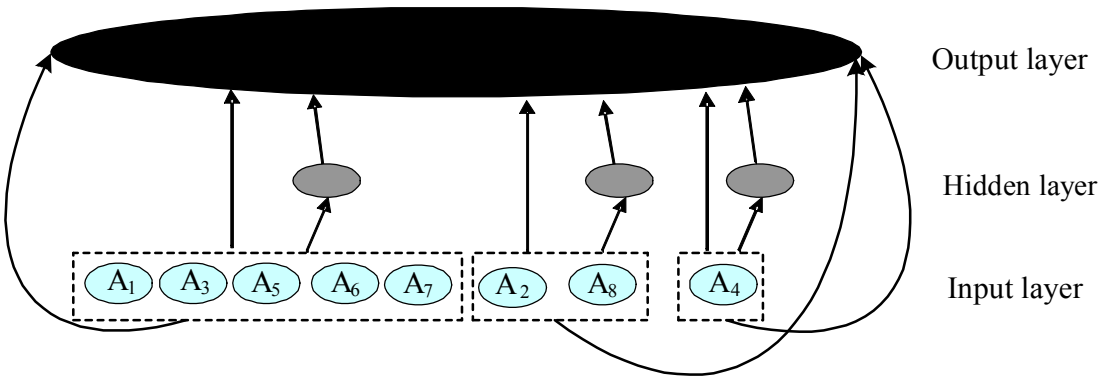


Figure 2.8: ILNNT1 for the diabetes problem

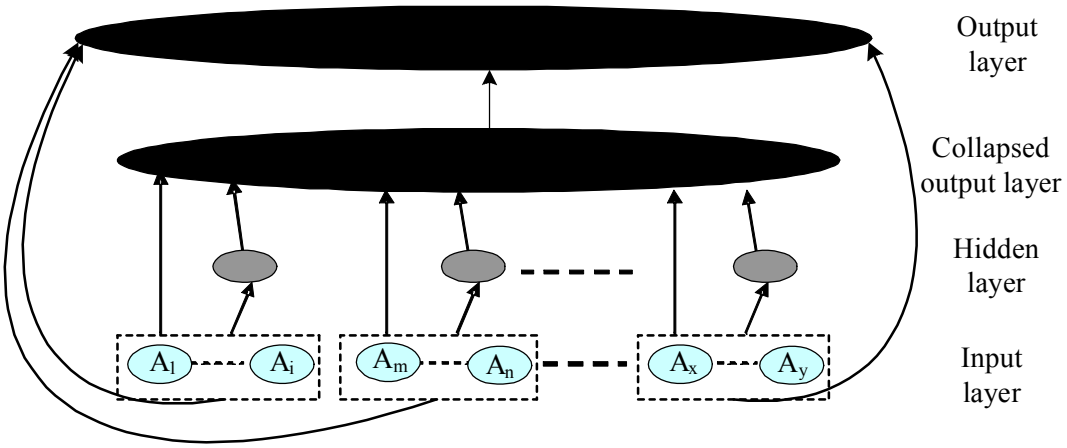


Figure 2.9: ILNNT2 (ILIA2) architecture

2.4 Experimental Setup and Data Sets

2.4.1 Experimental Setup

The NN were randomly initialized with values [-0.25…0.25]. 50% of the total examples in each data set is used as training data, 25% as validation data and the remaining 25% as testing data.

The method of early stopping using a validation data set is used to prevent overfitting. The network that obtains the lowest validation data set error will be

applied on the testing data. The testing data is not used in any training or partitioning process but presented to the trained NN to evaluate its generalization accuracy.

2.4.2 Data Sets

The data sets chosen are the diabetes, heart, glass and soybean data sets. The diabetes and heart data sets are binary class problems whereas the glass and soybean are multi-class problems. The soybean problem is a much larger data set in terms of higher number of input attributes and output classes compared to the other problems. This data set serves as an additional indicator to ILNNT performance for data sets with larger dimension. These data sets are chosen such that they are spread across different domains, i.e., from medical [11][70][71][121][127], agricultural [45][138] to criminal investigation [120][178], as such, this would show how well ILNNT performs generally in different fields rather than just in a particular field.

2.5 Experimental Results and Analysis

This section presents the results and findings of the ILNNT.

2.5.1 Interference and Partitioning

Examples of the interference table and resulting partitioning of the different data sets are given in this section.

Diabetes Data Set: The groupings resulting from the partitioning algorithm using different starting attributes are shown in Table 2.4.

Table 2.4: Partitioning obtained - diabetes

Starting attribute used by algorithm	Resulting attribute grouping
1, 2	(1 2 3 5 6) (4 7) (8)
3, 4, 7	(1 3 4 7) (2 5 6) (8) (1 3 4 7) (2 6 8) (5)
5, 6	(1 3 5 6 7) (2 8) (4)
8	(1 2 3 6 8) (4 7) (5) (1 2 3 6 8) (4) (5 7)

Heart Data Set: An example of the interference table shows that attribute 6 (fasting blood sugar) interferes with most of the other attributes. As the other attributes have no interference among each other, there are only two possible partitions.

Table 2.5: Interference table - heart

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	23.04												
2	21.77	23.36											
3	13.95	14.68	14.68										
4	22.74	22.50	14.13	23.42									
5	22.83	22.25	14.24	22.95	23.86								
6	23.42	23.33	15.07	24.07	24.22	24.49							
7	22.07	21.59	14.09	22.47	22.60	23.31	22.94						
8	20.66	20.63	14.05	20.28	20.64	21.41	20.53	20.95					
9	17.41	18.26	12.95	18.04	18.08	18.68	17.50	17.11	18.45				
10	16.98	17.10	11.78	17.0	17.04	17.79	17.00	16.28	14.51	17.38			
11	18.94	18.66	11.97	19.21	19.25	19.41	19.53	18.44	15.84	16.40	19.73		
12	17.40	16.48	11.47	16.90	17.36	17.58	17.08	16.37	14.22	13.71	14.49	17.50	
13	18.21	18.49	12.97	18.74	18.62	18.73	17.35	16.77	16.18	14.92	16.19	14.59	18.88

Table 2.6: Summary of the interference table - heart

Attribute	Do not interfere with	Interfere with
1	2 3 4 5 7 8 9 10 11 12 13	6
2	1 3 4 5 6 7 8 9 10 11 12 13	-
3	1 2 4 5 7 8 9 10 11 12 13	6
4	1 2 3 5 7 8 9 10 11 12 13	6
5	1 2 3 4 7 8 9 10 11 12 13	6
6	2 11 13	1 3 4 5 7 8 9 10 12
7	1 2 3 4 5 8 9 10 11 12 13	6
8	1 2 3 4 5 7 9 10 11 12 13	6
9	1 2 3 4 5 7 8 10 11 12 13	6
10	1 2 3 4 5 7 8 9 11 12 13	6
11	1 2 3 4 5 6 7 8 9 10 12 13	-
12	1 2 3 4 5 7 8 9 10 11 13	6
13	1 2 3 4 5 6 7 8 9 10 11 12	-

Table 2.7: Partitioning obtained - heart

Starting attribute used by algorithm	Resulting attribute grouping
1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13	(1 2 3 4 5 7 8 9 10 11 12 13) (6)
6	(1 3 4 5 7 8 9 10 12) (2 6 11 13)

Glass Data Set: There are several interference detected among the attributes. A summary of attribute interference is presented in Table 2.9 and the resulting partitions that uses different starting attribute are shown in Table 2.10. With several interference detected, the algorithm gave a few grouping for the input attributes.

Table 2.8: Interference table – glass

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	**12.273**								
2	10.638	**11.043**							
3	9.567	9.428	**11.858**						
4	10.581	9.985	9.511	**10.543**					
5	12.063	10.535	10.053	10.540	**12.270**				
6	10.409	10.897	9.887	10.257	10.949	**11.328**			
7	11.248	10.450	9.033	10.110	11.936	10.763	**11.858**		
8	9.958	10.023	9.170	9.527	9.905	9.635	9.305	**9.994**	
9	12.129	10.889	10.267	10.399	12.241	11.423	11.893	10.091	**12.262**

Table 2.9: Summary of the interference table – glass

Attribute	Does not interfere with	Interfere with
1	2 3 5 6 7 8 9	4
2	1 3 4 5 6 7 9	8
3	1 2 4 5 6 7 8 9	-
4	2 3 5 6 7 8 9	1
5	1 2 3 4 6 8 9	7
6	1 2 3 4 5 7 8	9
7	1 2 3 4 6 8	5 9
8	1 3 4 5 6 7	2 9
9	1 2 3 4 5	6 7 8

Table 2.10: Partitioning obtained - glass

Starting attribute used by algorithm	Resulting attribute grouping
1	(1 2 3 5 6) (4 7 8) (9)
2	(1 7 8) (2 3 4 5 6) (9)
3, 4	(1 2 7) (3 4 5 6 8) (9) (1 2 9) (3 4 5 6 8) (7)
5, 8	(1 3 5 6 8) (2 4 7) (9) (1 3 5 6 8) (2 4 9) (7)
6, 7	(1 3 6 7 8) (2 4 5 9)
9	(1 2 3 5 9) (4 6 7 8)

Soybean Data Set: As the soybean interference table is too large to be presented here, only a summary of the interference table and the partitioning obtained are shown.

Table 2.11: Summary of the interference table – soybean

Attribute	Does not interfere with	Interfere with
1-4, 6, 11-14, 16-19, 21-35	1 – 35	-
5	1-7, 9, 11-35	8 10
7	1-14, 16-35	15
8	1-4, 6-9, 11-35	5 10
9	1-9, 11-35	10
10	1-4, 6-7, 10-19, 21-35	5 8 9 20
15	1-6, 8-35	7
20	1-9, 11-35	10

Table 2.12: Partitioning obtained - soybean

Starting attribute used by algorithm	Resulting attribute grouping
1-5, 16-35	(1-7, 9, 11-14, 16-35) (8 15) (10) (1-7, 9, 11-14, 16-35) (8) (10 15)
6-7	(1-4, 6-9, 11-14, 16-35) (5 15) (10) (1-4, 6-9, 11-14, 16-35) (10 15) (5)
8	(1-4, 6, 8-9, 11-35) (5 7) (10) (1-4, 6, 8-9, 11-35) (5) (7 10)
9, 11-15	(1-6, 9, 11-35) (7 8) (10) (1-6, 9, 11-35) (7 10) (8)
10	(1-4, 6, 10-19, 21-35) (5 7 9 20) (8) (1-4, 6, 10-19, 21-35) (7 8 9 20) (5)

2.5.2 Results Comparison

Comparisons are made with conventional NN training, i.e., all the input attributes are introduced together at the same time into the NN for training. In order for the basis of comparison to be consistent, both the conventional method and the ILNNT use the CBP learning algorithm [93] for network learning.

The results of the final architecture of ILNNT is derived after several stages like the formulation of individual discrimination ability, interference of any two attributes, the partitioning algorithm, etc. Therefore, the computational complexity of the algorithm would only be accurate if all the preprocessing steps are included in the calculation. However, it is hard to track the computational complexity of all these preprocessing steps, e.g., it is hard to quantify the computation of the interference table and the partitioning process. In addition, as classification is commonly done offline rather than online, only the generalization accuracy acts as the major evaluation metric in this chapter.

2.5.2.1 Diabetes Data Set

Table 2.13: ILNNT results comparison with the conventional algorithm – diabetes

Algorithm	Mean error (%)	Maximum error (%)	Minimum error (%)	SD
ILNNT	23.83	29.69	18.75	0.959
Conventional	24.32	27.08	19.27	1.05

The mean error of ILNNT (23.83%) is lower than the conventional method (24.32%). The lower mean error shows that it is important to have partitioning in the input space for problems with interference among the input attributes. In addition, ILNNT has a smaller standard deviation. One would like to have a small standard deviation so as to achieve consistent and reliable results.

2.5.2.2 Heart Data Set

The ILNNT has achieved a lower mean error on the heart data set as compared to the conventional method. Partitioning is needed for this problem as the attributes

show signs of interference. The ILNNT also has lower standard deviation as compared to the conventional method.

Table 2.14: ILNNT results comparison with the conventional algorithm – heart

Algorithm	Mean error (%)	Maximum error (%)	Minimum error (%)	SD
ILNNT	20.99	26.32	15.79	1.64
Conventional	21.56	26.67	14.47	2.18

2.5.2.3 Glass Data Set

The mean error result obtained by ILNNT (35.71%) has outperformed the conventional method (37.10%). The ILNNT is also a more consistent algorithm as it has a smaller standard deviation. By lowering the interference, NN training is able to achieve better performance. The conventional method could not handle a multi-class problem like the glass problem with many interfering attributes.

Table 2.15: ILNNT results comparison with the conventional algorithm – glass

Algorithm	Mean error (%)	Maximum error (%)	Minimum error (%)	SD
ILNNT	35.71	44.44	27.78	3.66
Conventional	37.10	47.17	24.53	4.54

2.5.2.4 Soybean Data Set

ILNNT mean error is about 11% (((7.71-6.86)/7.71)x100%) lower than the conventional method. The maximum and minimum errors are also lower. ILNNT is able to perform well for data sets with high input and output dimensions. Among all data sets, ILNNT has the best performance for the soybean data set in terms of percentage improvement over conventional method.

Table 2.16: ILNNT results comparison with the conventional algorithm – soybean

Algorithm	Mean error (%)	Maximum error (%)	Minimum error (%)	SD
ILNNT	6.86	11.18	2.34	1.00
Conventional	7.71	11.77	2.92	1.10

2.5.3 *T*-Test

The *t*-test [131][162] has been used as an additional indicator for the performance of the algorithms besides just comparing the mean results and standard deviations. The *t*-test is used when there are some relationships between members of each sample. The *P*-values against the conventional method for all data sets show significance, meaning that ILNNT has outperformed the conventional algorithm with confidence.

Table 2.17: Results of *t*-test

	Diabetes	Heart	Glass	Soybean
P-values	0.017674	0.046062	0.044871	2.56×10^{-6}

2.6 Conclusions

An effective way of grouping the input space of NN to reduce interference among the attributes for NN training is being proposed and discussed in detail in this chapter. By separating and training interfering attributes under different sub-networks, interference among attributes is effectively reduced. In addition, those attributes that are beneficial to each other are grouped together; implicitly established when beneficial attributes are not taken out from the batch and by the recursive effort of the partitioning algorithm to group the remaining non-interfering attributes together after each fundamental round. From the experimental results, ILNNT is able to work well

for problems with varying degrees of attribute interference. NN are able to perform better if the correct partitioning and grouping methods are applied.

Chapter 3

Training Neural Networks using Growth Probability-Based Evolution

Gradient descent-based learning algorithms like the Backpropagation (BP) [27][29][57][114][136][149] for NN training require the number of hidden neurons to be predefined and fixed throughout the training process. This prevents the structure of the network from adapting to the problem during learning. In many instances, some previous information and knowledge are necessary or a trial and error process is carried out. The former method is not suitable when a new task is given and the latter method is time consuming and tedious. Adaptive and automated algorithms are needed to address these issues. These concerns have been looked into by several researchers and among them the more significant ones are the constructive learning algorithms. There are several well-known constructive learning algorithms for the NN, i.e., Dynamic Node Creation (DNC) [10], Cascade-Correlation (CC) Algorithm [46] and Constructive Backpropagation (CBP) [93] which is inspired by CC. In CBP, a network is initialized with direct links from the input units to the output units. Hidden neurons are then added one at a time until the overall stopping criterion is met. When a new hidden neuron is added to the network, previous trained weights are freezed. Recently, there are attempts of building constructive one-hidden-layer NN based on using different activation functions for each hidden neuron within the networks [105] and having separate training for the input side and output side within the incremental constructive NN [104]. As the constructive methods use gradient-based approaches,

they are restricted to using neurons with differentiable functions. One of the advantages of hybridizing EA and NN is being able to use neurons which are non-differentiable. In addition, as EA possess global search capability, Evolutionary Neural Networks (ENN) [13][169] are likely to find an optimum or near optimum solutions. However, one of the drawbacks of EA approach is the high computational complexity since it is a population-based method, where a population of individuals is being initialized and evaluated at each generation. As compared to EA approach, gradient-based methods, like the conjugate gradient, Levenberg-Marquardt, etc., [15][16][69][150] are generally less computational exhaustive as they are single point techniques.

Due to the great potential of hybridizing EA and NN, this area of research has attracted the interests of many researchers. There are several studies on how EA can be used to train the weights of NN, optimize NN structure, combine with local optimizers and to select salient features for NN training [23]. The use of EA includes evolution strategies, genetic and evolution programming, and the simplest and more commonly used genetic algorithms. Palmes et al. [125] uses mutation strategies to address the issue of high computer-intensive operation of using gradient descent-based BP in ENN. Yao and Liu [189] use evolutionary programming to evolve both the weights and the architecture of NN. Abbass [2] illustrates an EA that optimizes the architecture of NN consisting of one hidden layer. It uses a vector which acts as a switch to represent whether a hidden neuron is active. While the maximum number of hidden neurons that can exist is predefined, the number of activated hidden neurons decreases or increases.

Verma and Ghosh [176] use genetic algorithms and linear least square method to train the hidden layer and the output layer, respectively. The training process starts

with a small number of hidden neurons and at the end of each training round, one hidden neuron is added and tested again. In this way, the number of hidden neurons is found by consistently referring to the evaluation metric at each round. In [191], while there are unresolved regions defined by the training examples, one hidden neuron (generated using genetic search) is added to the hidden layer. This is repeated until all regions defined by the training examples are resolved. Leung et al. [95] use GA to tune the network structure and parameters. Hidden neurons are incremented manually from a small number till the fitness criterion is satisfied.

Though there are several works on ENN, this chapter serves to provide a new perspective of hybridizing NN with EA in particular for classification. Classification is an important task in many domains and though there are several methods that can be used to find the relationship between the input and output space [3], among the different works, ENN stand out as one of the most promising methods. Earlier works used EA solely to train the weights of NN, and later works allow the structure to vary as the number of hidden neurons increases or decreases while the maximum number of hidden neurons is fixed (this is restrictive and might create a problem if the required number of neurons is larger than the maximum allowed).

The algorithm proposed in this chapter evolves both the structure and weights simultaneously and places no restrictions on the number of hidden neurons that can be encoded by the chromosome while overfitting is solved by using a validation data set. As compared to recent works, the algorithm uses a new operator, i.e., growth probability (P_g) to determine whether hidden neurons should be increased. Overall the training process is made simpler and more elegant without the need of numerous checks and many predefined constraints. Presented works in literature often require several operators to vary the connections and neurons. Different operators are

typically used to insert or delete a connection or a neuron, and doing so requires numerous checks on the operability of the networks for broken links or dangling neurons. In addition, this chapter also managed to evolve NN that is able to obtain competitive classification accuracy while keeping the network complexity low.

Two versions of training algorithms are proposed in this chapter. The first algorithm evolves NN using growth probability-based evolutionary technique (NN-GP) and applies the same P_g on the whole population, i.e., the probability of increasing the number of hidden neurons for each NN in the population is the same. Given a P_g, ($P_g \bullet 100$)% of the chromosomes (a representation of the number of hidden neurons and connection weights) in the offspring population will undergo growing in terms of lengthening, thus representing more hidden neurons. The second algorithm presented in this chapter is the self-adaptive version of NN-GP (NN-SAGP), which evolves the parameter P_g together with the NN architecture and weights. A NN is allowed to increase its number of hidden neurons depending on own P_g.

The algorithms proposed in this chapter are evaluated based on several criteria, namely, the classification accuracy on training data set, generalization accuracy, computational complexity of evolved networks and the training time. The objectives of the algorithms are to evolve a network with high accuracy and with optimal architecture.

This chapter is divided into 6 sections. The next section illustrates the NN that is being modeled and the stopping criterion used. In Section 3.2, an overview of the NN-GP with detailed descriptions of the various operators will be explained. Section 3.3 gives the NN-SAGP algorithm. The experimental setup and data sets used are presented in Section 3.4. Results of experiments using several benchmark problems

are presented, analyzed and compared with other existing works in literature in Section 3.5. The conclusions for this research are given in Section 3.6.

3.1 Neural Network Modeled and Stopping Criterion

The architecture of the Multi-Layer Perceptrons (MLP) which is being evolved and the stopping criterion used are presented in the following sections.

3.1.1 Neural Network Architecture

The NN architecture modeled by the growth probability-based evolutionary technique is a fully connected NN consisting of an input layer, a hidden layer and an output layer. As it is well-known that a single hidden layer is sufficient for a MLP to compute a uniform approximation for any given training data (universal approximation theorem) [72], the number of hidden layers is fixed at one. Using more hidden layers increases the network complexity, however does not guarantee better results.

Given an input-output data set (x_i^m, y_i^o) where $x_i \in \Re^m$, m is the number of inputs, i={1,2,…..,N}, N is the number of patterns in the data set, $y_i \in \Re^o$, o is the number of outputs, the input layer will consist of m number of input units each representing the individual input from the data set. The output layer will consist of o number of output neurons which represent the outputs. All bias input values are set to 1. The hidden neurons use sigmodal activation function while the output neurons use linear summing function. Determination of output class for a presented input pattern is decided by the winner-take-all strategy, i.e., decided class by the network is the corresponding output neuron with the highest output value. The NN modeled is shown in Figure 3.1.

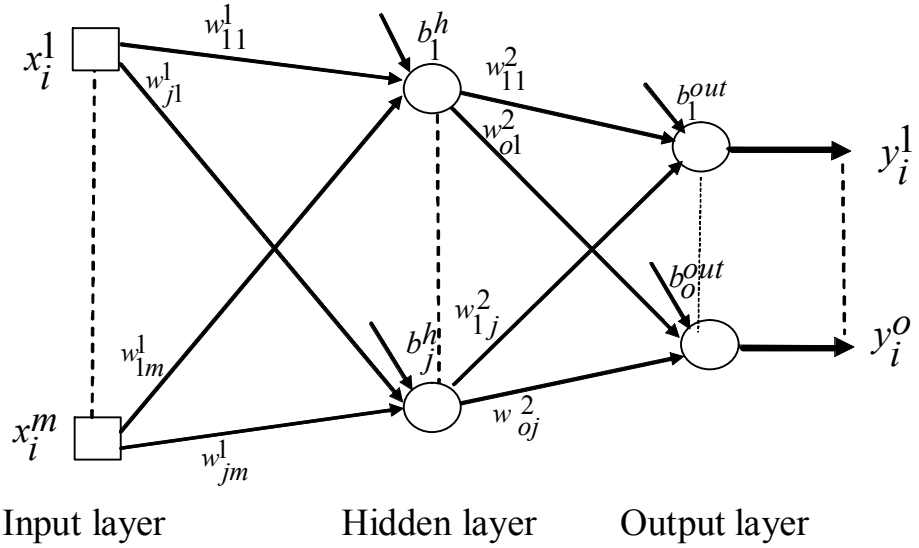


Figure 3.1: Neural network modeled

where w^1_{jm} is the weight for connection between the *m*th input unit and the *j*th hidden neuron, b^h_j is bias weight of the *j*th hidden neuron, w^2_{oj} is the weight for the connection between the *j*th hidden neuron and the *o*th output unit and b^{out}_o is bias weight of the *o*th output unit.

3.1.2 Overall Stopping Criterion

Neural networks that are under-trained will not be able to learn the problem well. On the other hand, over-training might cause overfitting (a network with larger training error could be better than one with lower error as the latter has concentrated on the peculiarities while losing the regularities needed for good generalization accuracy [56]). One of the well-known methods in addressing this issue is to use a validation data set. The validation data set is used as a pseudo testing data set. Training stops when maximum validation accuracy (minimum validation error) is reached and the current network state is used on the testing data set. However, as

there are many local optima in the validation data set, there are some issues when using it. Firstly, during the initial phase of training, the error on validation data set will be oscillatory and secondly, in order to recognize an optimum, training has to proceed until the accuracy decreases [48][92][117][130]. Hence, the overall stopping criterion is defined as follow:

To avoid the initial oscillation phase, Condition 3.1 has to be satisfied before proceeding to next stage.

$$\frac{1}{t}\left(\sum_{i=g-t+1}^{g} A_t(i) - \sum_{i=g-2t+1}^{g-t} A_t(i)\right) < TS \qquad \text{Condition 3.1}$$

where g is the current generation number, t = training generation length (number of generations over which training accuracy is averaged), $A_t(i)$ = average training accuracy of parent population and TS = training stop threshold. When Condition 3.1 is fulfilled, testing for Condition 3.2 is carried out. Accuracy on the validation data set is measured every generation and training is stopped as soon as Condition 3.2 is fulfilled. Generalization accuracy is then computed for that state of network which has the highest validation accuracy.

$$A_v(i') - A_{av} > 0, \qquad \text{Condition 3.2}$$

$$A_{av} = \frac{1}{v}\sum A_v(i), \forall i \in [i'+1, i'+v]$$

where i' is the generation that obtain the highest validation accuracy, A_{av} = average validation accuracy of parent population and v = validation generation length (number of subsequent generations compared to the generation with highest accuracy).

3.2 Growth Probability-Based Neural Networks Evolution

Without any growing phase, the network is not able to expand and will be stagnant. Fixing the number of hidden neurons in advance without giving the network an opportunity to find the appropriate number of neurons is not recommended. It is thus necessary for an algorithm to be in place to grow the number of hidden neurons in order to obtain a network with appropriate size.

3.2.1 Overview

In this sub-section, an overview of the algorithm (NN-GP) is described using a step by step illustration. More detailed descriptions of the operators are given in the subsequent sub-sections.

Step1: Initialization

Initialize an initial chromosome population of size μ (parent population size) with each individual representing a NN with one hidden neuron. The networks in the parent population will represent a pool of individuals with high fitness at every generation. The genotype representation that is being initialized at the start of the evolutionary process is shown in Figure 3.2.

$$(b_1^{out}, b_2^{out}, \ldots, b_o^{out}, w_{11}^1, w_{12}^1, \ldots, w_{1m}^1, b_1^h, w_{11}^2, w_{21}^2, \ldots, w_{o1}^2)$$

Figure 3.2: A chromosome representing a neural network with one hidden neuron

b_o^{out} is the bias weight of the oth output unit. w_{1m}^1 is the weight for connection between the mth input unit and the first hidden neuron. b_1^h is the bias weight of the

first hidden neuron. w_{o1}^{2} is the weight for connection between the first hidden neuron and the *o*th output unit.

The weights are assigned random values in the range [-0.5,..,0.5]. The weights of output bias (b^{out}) are situated at the front end of the chromosome and this representation serves two purposes. Firstly, the growing process can be easily carried out as the addition of hidden neuron weights can be directly appended at the end of the chromosome string. Secondly, it minimizes interruption to the weights connected to the hidden neurons. This set of weights is not connected to the hidden neurons, but connected to the output units, thus should be separated from the rest of the weights in the chromosome.

Step 2: Selection of Parents for Offspring Creation

Selection of parents for offspring creation uses the tournament selection (size three and with replacement). In tournament selection, three NN are randomly selected from the parent population. The network with the highest fitness level, i.e., highest classification accuracy on the training data set (the validation data set should not be used to guide the improvement of the fitness of the chromosomes since it acts as a "pseudo testing data set"), will be taken as a snapshot for offspring creation process. The three chromosomes are then replaced back into the parent population. The selection of second parent follows the same procedure used to select the first parent. Fitness evaluation of a chromosome is given in Equation 3.1.

$$f_k = \frac{C_k}{N} \bullet 100\% \tag{3.1}$$

f_k is the fitness of chromosome k , k= (1, 2…L), L is the number of chromosomes to be evaluated. C_k is the number of correctly classified patterns by chromosome k and N is the total number of patterns in the training data set, i.e., ($C_k \leq N$).

Step 3: Offspring Creation

The two methods for offspring creation are either the crossover or replicate operation. The crossover and replication operations are mutually exclusive events. If the two selected parents are of the same length, crossover is done else replicate operation is carried out. Mutation is carried out only after crossover or replicate operation.

- **Crossover Operation**

If the two parent chromosomes selected are the same length, then crossover as described in Section 3.2.2 is applied.

- **Replicate Operation**

Since chromosomes grow at different rates, length of chromosomes in the population may differ. If the two parents chosen are different in length, they will be replicated or in another word 'cloned' to create the new offspring in this current generation. Each parent will replicate once, and correspondingly two offspring are created. The motivation behind this operation is because parents emerging from tournament selections are the fitter chromosomes compared to their peers and therefore worthy of 'cloning' which is a process done to replicate good individuals.

For different length chromosomes, crossover is still possible while maintaining workable NN (NN with no broken links or missing node). However, no crossover is done here as the architecture of each NN is to be preserved apart from being affected by the growth probability. In the study of growing the number of hidden neurons

using growth probability, disruption to the network architecture by other influences should be avoided.

Step 2 and step 3 are repeated until λ (children population size) offspring are created.

Step 4: Mutation

The probability that an allele of a chromosome will be mutated is based on the mutation probability, P_m. By using the mutation operator, the algorithm is able to explore new areas of search space not visited before. Mutation operator used is real-value mutation. A random value between [-0.2,...,0.2] is added to the allele. At any one time, if the value of the allele that undergoes mutation exceeds the range [-1, 1], boundary conditions are enforced as biases should not be created. Boundary conditions are given in Equation 3.2.

$$v_q = \begin{cases} 1, & \text{if } v_q > 1 \\ -1, & \text{if } v_q < -1 \end{cases} \tag{3.2}$$

where v_q = mutated value of allele occupying the qth position of the chromosome

Step 5: Growing Operation

This step is one of the key steps in structuring the architecture of the NN. The growing phase allows a chromosome to represent more hidden neurons. With more hidden neurons, the NN may be able to classify the training patterns better as more hyper-planes are formed to separate the different classes. However, it should be noted that a larger NN might perform well on the training data set but badly on the testing data set, in this case overfitting has occurred. The growing operation is described in detail in Section 3.2.3.

Step 6: Selection of Children

Selection of children is to bring forward children with better fitness in the current generation to the next generation to form the parent population. Selection of μ children as parents for next generation is based on selecting the best μ individuals from λ (children population size) offspring, i.e., (μ , λ) strategy. The best individuals are evaluated using the fitness function given in Equation 3.1.

Elitism (size 2) is also used in NN-GP. Step 2 to step 6 are repeated until the overall stopping criterion is satisfied.

3.2.2 Crossover Operator

The encoding method used for all the genotype representation is the same and it allows any point to be crossover while maintaining the structure of NN. In NN-GP, two crossover points are selected. First crossover point is situated where the encoding of the bias weights for the output neurons ends and the second crossover point is in the middle of the remaining string. The reason behind choosing the first point is the fact that the bias weights of the output neurons are not connected to the hidden neurons, hence alleles representing these biases are to be grouped, and should be separated from the rest of the alleles that represent weights of the hidden neurons. The second point is chosen to facilitate exchange of useful structural neuronal information between the networks.

When two crossover points are selected, a parent chromosome is split into three sections. The crossover process is then carried out as follows; the first section of offspring 1 will carry genes from the first section of parent 1. The second section of this offspring will be determined by a random number, $x \in [0,1]$. If $x \geq 0.5$ the second section of parent 2 will be concatenated to offspring 1, else if $x < 0.5$ the second

section of parent 1 will be used. If the second section of offspring 1 comes from parent 1, the third section will be from parent 2. The same principles apply to the creation of offspring 2.

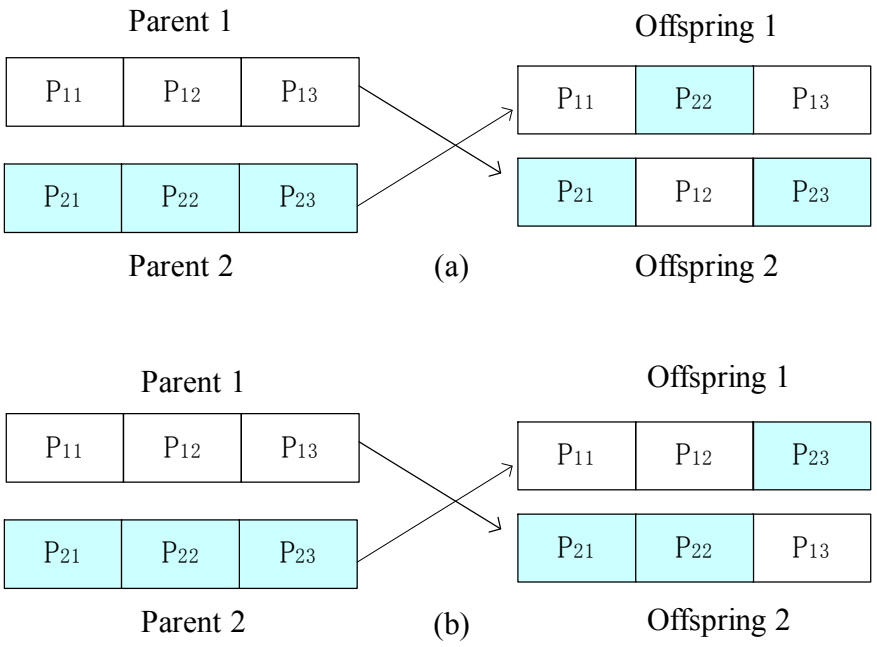


Figure 3.3: (a) Crossover operation for $x \geq 0.5$ (b) Crossover operation for $x < 0.5$

3.2.3 Growing Operator

The probability of growth is the core parameter used in the growing process of the NN. Motivation for this step is to increase the number of hidden neurons together with its associated connection weights concurrently and probabilistically. By using this operator, a NN is able to change its structure and hence might be able to improve its overall performance. It differentiates itself from the probability of crossover (P_c) and mutation (P_m) in the sense that, P_c governs the chance of any two existing networks coming together to form new structures while retaining full information from the old networks. No new information is introduced in this case. The mutation probability (P_m) governs the chance of mutation. Mutation is usually characterized by

small changes in the network structure and not by any drastic changes. If mutation is carried out with drastic changes, the whole process will emerge as random search for a NN that performs well. Random search does not guarantee optimal solutions within a reasonable time and in worst case, no solutions near optimal are ever found. On the other hand, P_g works on increasing the number of hidden neurons with its associated weights. The newly created NN has the new neurons and weights but at the same time retains its old neurons and weights.

In NN-GP, the value of P_g is fixed and all the NN depend on the fixed P_g to determine whether new neurons should be added at each generation. Since P_g is consistent and applied throughout the whole population, ($P_g \bullet \lambda$) number of children in a children population of size λ will have new hidden neurons added. There will be two groups of chromosomes within the children population. In one group, the number of hidden neurons is increased while the other group retains its existing network structure. Children that are fitter after growing will have higher chances to proceed to the next generation while those that turn weaker will have lower chances. If lengthening the chromosomes produces better fitness, the parent population in the next generation will have more chromosomes that are grown, representing more neurons than the parent population in the previous generation. If the lengthened (grown) children chromosomes do not have better fitness than those chromosomes that did not undergo growing, the parent population in the next generation will have an average population of chromosome length that is maintained at the same length as the parent population of the previous generation. The evolution principles together with the growing mechanism automatically determine the near optimal number of neurons needed for the NN.

It is normal that the growing of hidden neurons and connection weights might not be useful all the times and there could be instances that the growing will introduce noise. It is part and parcel of EA that noise are introduced in various manners, e.g., through mutation, crossover, etc. There would also be many instances that the growing process is very useful (newly explored good solution space or good solutions obtained by growing). Evolutionary techniques are more interested in these successful attempts than those unsuccessful attempts. The poorer networks will eventually be filtered out by the evolution process, while the better networks would be brought forward to later generations and made stronger.

The growth rate which determines how many neurons are to be added is determined by the Gaussian distribution. This process is described in detail in Section 3.2.4.

To illustrate the process of growing hidden neurons, Figure 3.4 shows how an extra hidden neuron can be added to a chromosome representing one hidden neuron in the genotype space. The newly created chromosome represents the weight values of a two hidden neuron network. The appending of the new alleles takes place at the end of the chromosome string. Appendages takes place in blocks, with each block representing one hidden neuron. Any number of blocks can be appended without disrupting those alleles that initially existed. In the phenotype space, another example illustrating the addition of two hidden neurons to an existing NN containing three hidden neurons is shown in Figure 3.5. Figure 3.6 gives the representation for an arbitrary length chromosome.

$$(b_1^{out}, b_2^{out}, \ldots, b_o^{out}, w_{11}^1, w_{12}^1, \ldots, w_{1m}^1, b_1^h, w_{11}^2, w_{21}^2, \ldots, w_{o1}^2, w_{21}^1, w_{22}^1, \ldots, w_{2m}^1, b_2^h, w_{12}^2, w_{22}^2, \ldots, w_{o2}^2)$$

← Existing alleles →← Newly created hidden neuron →

Figure 3.4: Growing of an addition hidden neuron

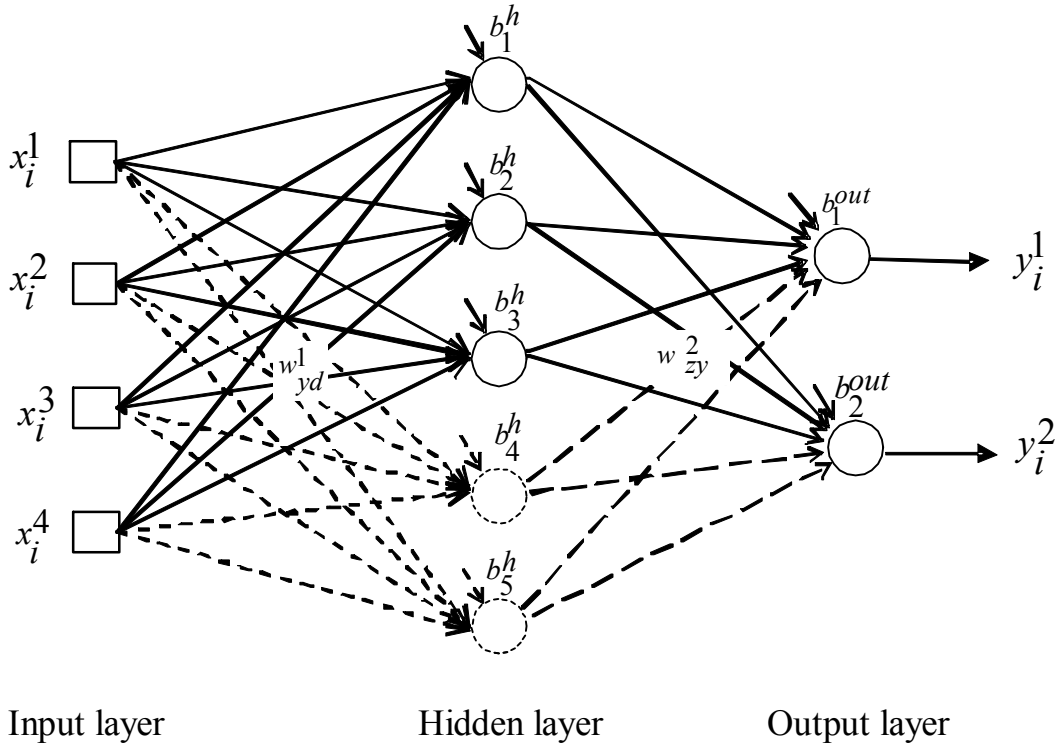


Figure 3.5: Addition of two hidden neurons to existing neural network structure

The dotted lines and neurons are newly introduced into the existing neural network structure which is represented by the solid lines. w_{yd}^1 represents weight connecting the *d*th input unit to the *y*th hidden neuron. w_{zy}^2 represents weight connecting the *y*th hidden neuron to the *z*th output unit.

$$(b_1^{out}, \ldots, b_o^{out}, w_{11}^1, \ldots, w_{1m}^1, b_1^h, w_{11}^2, \ldots, w_{o1}^2, w_{21}^1, \ldots, w_{2m}^1, b_2^h, w_{12}^2, \ldots, w_{o2}^2, \ldots\ldots, w_{j1}^1, \ldots, w_{jm}^1, b_j^h, w_{1j}^2, \ldots, w_{oj}^2)$$

Figure 3.6: General representation for an arbitrary length genotype representation

3.2.4 Determination of Growth Rate

Using Gaussian distribution [97][99] to determine the growth rate (number of neurons to be added each time) mimics the nature where creatures grow at different speeds, some faster while some slower but the majority growth rate still falls within a certain acceptable range. This is reflected by the Gaussian distribution graph which has higher distribution centered about the mean. To do this, a random number based on a Gaussian distribution is generated and depending on the range it falls in, the corresponding number of neurons is grown.

The optimal number of hidden neurons for a network might be a few neurons more than what it represents now. If this solution is far from its current location, NN has to increase the number of hidden neurons not just by one or two. Therefore insertion is not restricted to one hidden neuron at a time and by adding different number of hidden neurons, it provides a mechanism to move from a neighborhood search to explore a larger search space (global search), thus escaping from local optima.

It should be noted that growing too many neurons can have adverse effects on the networks as it might cause too much disruption to the network. Hence, the Gaussian distribution mean is set to 0 and variance 1. If the mean and variance is set too large, the network would be growing too many neurons at a time. The illustration on determination of growth rate is shown in Figure 3.7.

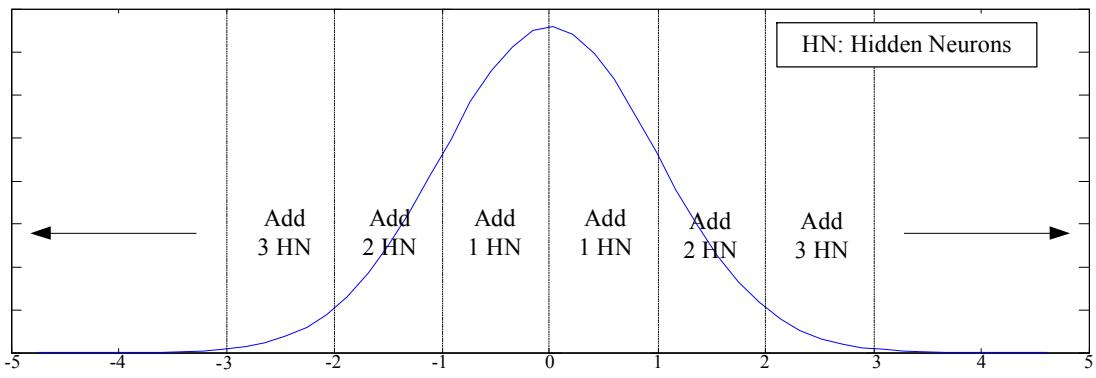


Figure 3.7: Growth rate based on Gaussian distribution

3.3 Self-Adaptive Growth Probability-Based Neural Networks Evolution

Both the NN-GP and NN-SAGP need to set parameters like the population size and mutation probability. In addition, the NN-GP needs to set the P_g. The main parameter P_g proposed here would naturally raise the question of what is a suitable value to be used for each specific problem. The earlier described NN-GP depends on a user input P_g and this value is fixed throughout the whole evolution process. Typically, different values for this parameter are tried before a suitable value for a particular problem is obtained. Self-adaptive growth probability-based NN evolution (NN-SAGP) is proposed here to address the aforementioned issue.

3.3.1 Probability of Growth

The probability of growth at the initial stage should be higher than the later stage of NN training. There is high pressure to grow the networks in order to increase the accuracy at the start of the training process. More initiated increase of neurons should be done at the initial stage to speed up the process of finding a near optimum solution. Not only a network that is appropriate in size is to be found, there is a need to evolve the network within a reasonable time too. An algorithm making large jumps in the search space at the beginning of the evolution process relates to having global search for suitable NN architectures and when it has reached a more steady position, it narrows down to fine-tuning and a more local search by using a smaller P_g.

The initial value of P_g is not carefully set but randomly generated for each individual in the population. This parameter is then evolved through the generations. In this case, there are some chromosomes with P_g greater than 0.5 and others that are

lower. Though P_g should be higher in the initial generations, there should not be too many chromosomes having a high value too. The population is to have a mix of chromosomes with different P_g. During training, P_g is naturally evolved to follow the need of growing hidden neurons.

3.3.2 Self-Adaptive Method

The general algorithm framework used to evolve NN in NN-SAGP is similar to NN-GP. The different steps between these two algorithms are highlighted below:

Step 1: An additional parameter (P_g) is assigned to each chromosome. In NN-SAGP, each network's chance of growing neurons depends on own P_g rather than the P_g that is used for the whole population.

Step 2: Same as NN-GP.

Step 3: Same as NN-GP.

Step 4: Same as NN-GP.

Step 5: This step is executed at every five generations instead of every generation. Networks with higher P_g will have larger probability of structural change compared to those networks with lower P_g. During the fitness evaluation process, those networks without any changes in their structure will have higher chances of survival since their entire network are trained as compared to those networks with newly grown neurons where their structures are only partially trained. This will have the effect of eliminating networks with high P_g during the evolution process and those networks with a low P_g will be able to survive better than those with higher P_g. By doing interval generation growing, networks are given a greater chance to learn and

stabilize their structure before the next growing cycle starts. For the NN-GP, this problem does not exist since all networks depend on the same P_g value.

Step 6: Same as NN-GP.

Before the execution of the growing process, the parameter P_g is mutated. Mutation takes the form of real-value mutation where a value between [-0.2,…,0.2] is randomly generated and added to the P_g. Mutation is done before the growing and fitness evaluation process. The rationale behind this ordering is based on the principles of evolution strategies. The primary evaluation states that a NN after growing is good if it performs well for classification. The secondary evaluation states that the parameter P_g is good if the NN it created performs well.

3.4 Experimental Setup and Data Sets

The three data sets used are the cancer, the diabetes and the card data sets. The parent population size (μ), children population size (λ), mutation probability (P_m) and tournament size used in NN-GP are 25, 150, 0.15 and 3, respectively. The growth probabilities used are 0.2, 0.4, 0.6 and 0.8. These parameters settings are the same for NN-SAGP except the growth probability is not fixed. Results for mutation probability (P_m) = 0.3 is also presented for NN-SAGP. The stopping criterion parameters are t (training generation length) = 7, TS (training stop threshold) = 0.1 and v (validation generation length) = 7. The parameters for Gaussian distribution are mean = 0, variance (σ^2) and standard deviation (σ) = 1.

3.5 Experimental Results and Analysis

All algorithms are evaluated based on classification accuracy on the training data set, generalization accuracy on the testing data set and the complexity of the trained NN. Comparisons are made with NN trained by normal EA without growth probability (EANN) and gradient descent-based backpropagation (BPNN). The EANN and BPNN have fixed architecture with different number of hidden neurons. For fair comparison, the settings used are as explained below:

- The parent population size (μ), children population size (λ), mutation probability (P_m) and tournament size used in EANN and NN-GP/SAGP (P_m = 0.15) are the same. The representation and variation operators are also similar for both EANN and NN-GP/SAGP.
- The number of generations for EANN is taken to be estimated around the average number of generations used for NN-GP ($P_g = 0.2 \text{ to } 0.8$) and NN-SAGP ($P_m = 0.15 \text{ and } 0.3$). Since BPNN is not generation-based, but depends on the number of epochs (one evaluation for each epoch), the number of evaluations (stopping criterion) used for BPNN is estimated to be around the average number of evaluations used for NN-GP/SAGP.
- The NN architecture for BPNN and EANN is similar to NN-GP/SAGP, i.e., only a single hidden layer is used.

In addition, other relevant existing works in literature that managed to get good generalization results are also compared. The comparisons are not intended to be exhaustive. Instead, they are used as a guide to assess how the proposed algorithms perform. The experimental results are also analyzed statistically.

The actual training time of NN is highly dependent on the programming language, efficiency of the code, speed of the computer, etc. Apart from the training time being presented, the number of evaluations (based on the number of times the training data set is being evaluated) which gives a more accurate measurement of computational complexity of the proposed algorithms is also given.

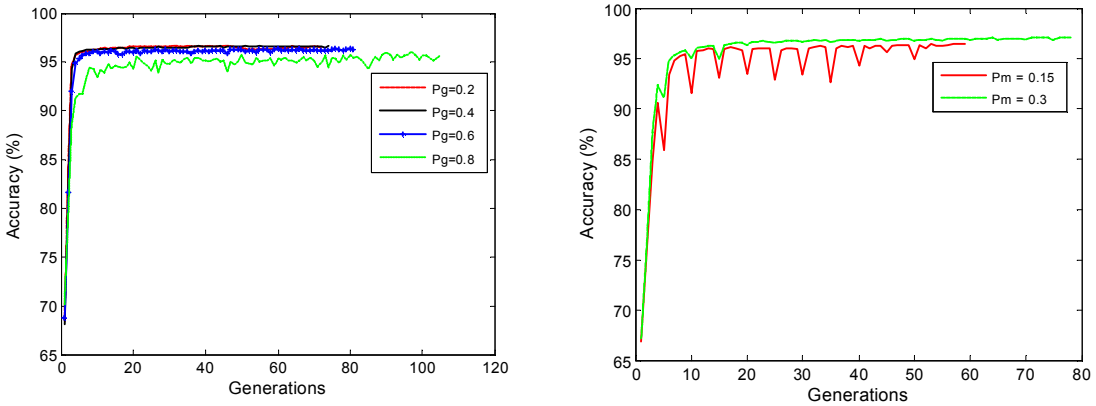
3.5.1 Cancer Problem

The results of NN-GP and NN-SAGP on the cancer problem are presented in the following sub-sections.

3.5.1.1 Results on Training Data Set

NN-GP: During the initial stage, the average classification accuracy of the parent population increases as the number of generations increases. This increase is slow after the fifth generation. Populations using P_g = 0.8 converges slightly slower and is more oscillatory. When P_g is large, a large number of networks is being grown. Searching in the objective space is now longer and leads to the slower convergence rate. Overall, the graph presented has a relatively smooth surface rather than a fluctuating one which shows NN evolved using growth probability are stable in improving the classification accuracy.

NN-SAGP: The classification accuracy shows an overall increasing trend. The increase is rapid during the first few generations and converges after that (these observations were consistent with NN-GP trends). The dip at every five generations is due to the growing of neurons. When there is a new neuron, this new neuron which has not been trained leads to a decrease in accuracy. After a training round, accuracy increases again.



(a) (b)

Figure 3.8: Classification accuracy on cancer training data set using (a) NN-GP (b) NN-SAGP

The value of P_g fluctuates about the value 0.5 for the first few generations and followed by a decreasing trend. This could relate to the NN training which shows that during the initial generations, the classification accuracy is low and hence there is a pressure to increase the number of hidden neurons for better classification accuracy. After some generations, it becomes difficult to increase classification accuracy even with more hyper-planes, thus P_g decreases steadily. Eventually, classification accuracy has converged and the parameter P_g starts to decline.

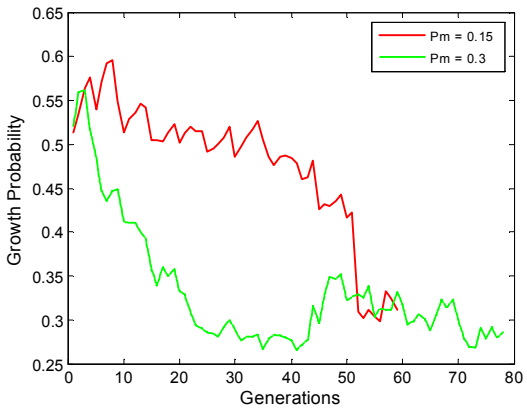
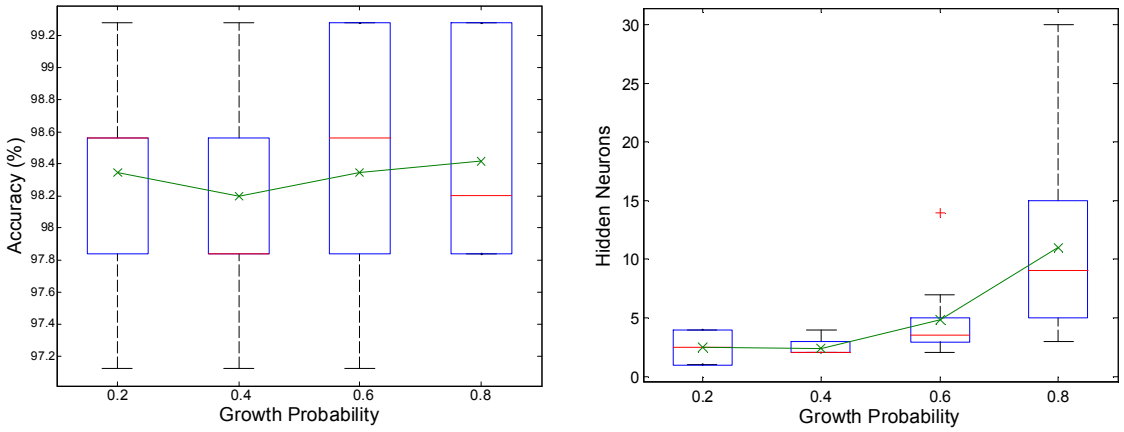


Figure 3.9: Value of growth probability over the generations for cancer training data set (NN-SAGP)

3.5.1.2 Different Values of Growth Probability on Testing Data Set

NN-GP: The best mean generalization accuracy is achieved when $P_g = 0.8$ and worst mean accuracy is obtained if a $P_g = 0.4$ is used. Though, $P_g = 0.8$ has the highest accuracy it requires a large number of hidden neurons, thus using $P_g = 0.2$ would be a better choice (with a small trade-off in accuracy, the number of neurons is greatly decreased). The box plots are also shown. The median (accuracy) for $P_g = 0.2$ (at the upper quartile) is also higher than the median for $P_g = 0.8$.



(a) (b)

Figure 3.10: (a) Classification accuracy on cancer testing data set (b) No. of neurons used by the networks – (NN-GP)

The numbers in Table 3.1 are the average for the best performing NN of each training run and applied on the testing data set. As P_g increases the time taken also increases and this could be explained by the observations made earlier. As P_g increases, the number of hidden neurons in a network increases, thus with larger number of hidden neurons, complexity of networks increases and correspondingly the computational time and evaluations are higher.

It has been noted that the time taken for NN-SAGP to evolve the NN are 218.58s and 656.71s when using P_m = 0.15 and 0.3, respectively, while the average number of generations are 43.6 and 56 for P_m = 0.15 and 0.3, respectively.

Table 3.1: Accuracy, time taken, number of generations and evaluations - cancer

Cancer	Probability of growth			
	0.2	0.4	0.6	0.8
Accuracy (%)	98.35	98.20	98.35	98.42
Time (s)	249	366.2	529.4	1404.4
Generations	48.5	46.6	59.4	72.3
Evaluations	8488	8155	10395	12653

3.5.1.3 Comparison

Apart from EANN and BPNN, results of NN-GP and NN-SAGP are compared to Mutation-based Genetic Neural Network (MGNN) [125], Memetic Pareto Artificial Neural Network (MPANN) and SPANN [2]. MGNN uses evolutionary programming for weights learning. MPANN uses EA to optimize both the weights and architecture of the NN while SPANN is the self-adaptive version of MPANN.

Table 3.2: Comparison of results - cancer

Algorithm	Test accuracy (%)	No. of hidden neurons
NN-GP (P_g = 0.2)	98.35 ± 0.5923	2.5 ± 1.2693
NN-SAGP(P_m = 0.15)	98.35 ± 0.6825	1.1 ± 0.3162
NN-SAGP(P_m = 0.3)	98.42 ± 0.8168	1.70 ± 0.8233
MGNN	96.86	-
MPANN	98.1 ± 0.5	4.125 ± 1.360
SPANN	98.3 ± 4.60	5.933 ± 0.961
EANN1	97.93 ± 0.9788	2
EANN2	98.07 ± 1.0675	7
BPNN1	97.07 ± 1.324	2
BPNN2	96.5 ± 1.595	7

The results presented in Table 3.2 show that NN-GP is able to obtain 98.35% generalization accuracy using P_g = 0.2 and NN-SAGP has obtained 98.42% generalization accuracy. Both have outperformed all other algorithms in this aspect. The smallest network evolved is by NN-SAGP giving 1.1 hidden neurons, which is about 73% and 81% less than what MPANN and SPANN used, respectively. Compared to EANN and BPNN, NN-GP and NN-SAGP have higher accuracy with lower standard deviation. The algorithms proposed are efficient and effective for the cancer problem giving high classification accuracy on unseen data while using small NN sizes.

Paired *t*-test has been performed using the best performing P_g parameter setting and the best NN-SAGP result against all other settings. The *P*-values show close match among NN-GP and NN-SAGP results suggesting there are a few good solutions in the search space and the proposed algorithms are able to find them. NN-GP and NN-SAGP have also managed to outperform the BPNN with confidence.

Table 3.3: *P*-values of the paired *t*-test - cancer

NN-GP				NN-SAGP		EANN		BPNN	
0.2	0.4	0.6	0.8	0.15	0.3	2HN	7HN	2HN	7HN
-	0.3392	0.5000	0.3988	0.4999	0.4110	0.1364	0.2093	0.0184	0.0036
0.4110	0.2480	0.4266	0.5	0.4158	-	0.1183	0.2139	0.0197	0.0066

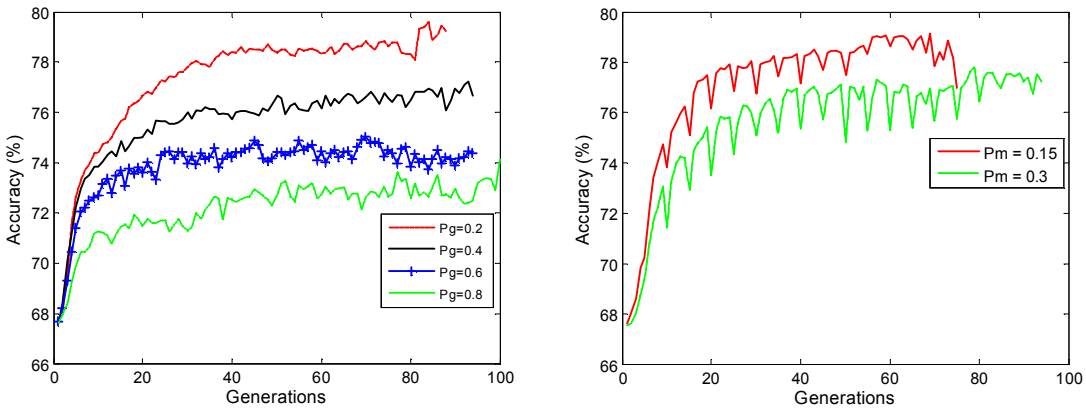
3.5.2 Diabetes Problem

The results of NN-GP and NN-SAGP on the diabetes problem are presented in the following sub-sections.

3.5.2.1 Results on Training Data Set

NN-GP: The best performance is obtained using P_g = 0.2. The performance on training data set deteriorates as P_g increases. The graph presented a rather smooth plot for lower P_g and a fluctuating one for larger P_g. More NN are being grown for larger P_g, thus greater diversity in structures among the networks of the population.

NN-SAGP: It seems that the results on classification accuracy are rather fluctuating though the overall trend shows increasing classification accuracy. Similar to the cancer problem, there is a dip in accuracy every five generations due to the insertion of new neurons.



(a) (b)

Figure 3.11: Classification accuracy on diabetes training data set using (a) NN-GP (b) NN-SAGP

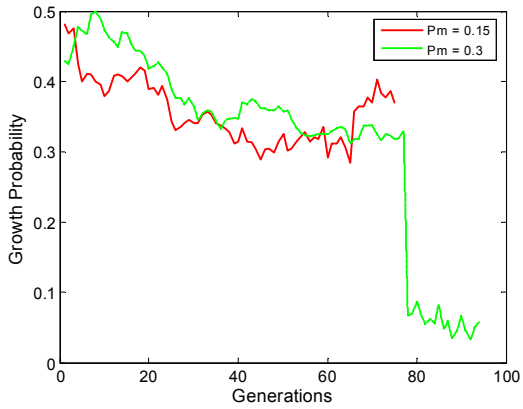
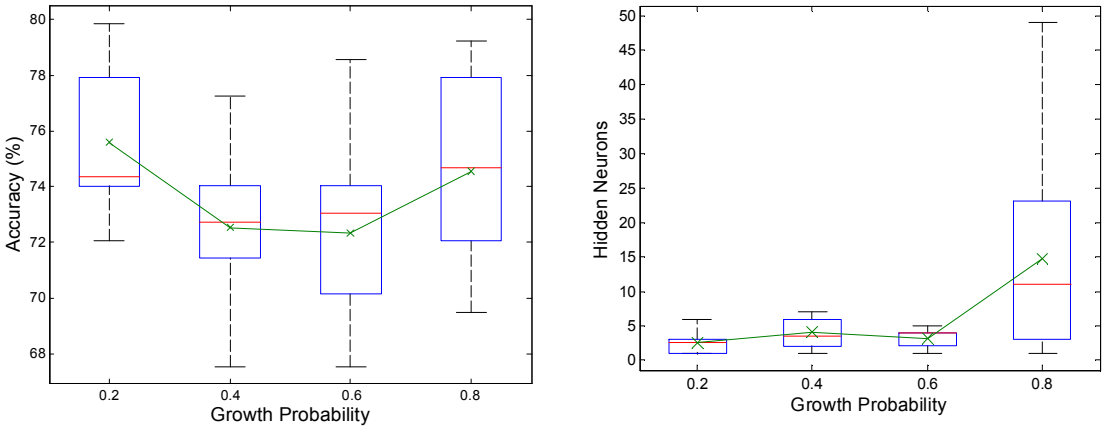


Figure 3.12: Value of growth probability over the generations for diabetes training data set (NN-SAGP)

3.5.2.2 Different Values of Growth Probability on Testing Data Set

The best performance for mean accuracy on the testing data set is obtained using P_g = 0.2. Using larger number of hidden neurons does not necessary mean an increase in generalization accuracy.

From Figure 3.13b and Table 3.4, it can be deduced that the probability of growth has greater influence on the number of hidden neurons while the number of generations used has negligible effect. If the number of neurons is to be pegged to the generation number, it would most probably be a number that is consistent for all P_g. This is noteworthy because NN that increases hidden neurons based on generation number will cause the number of hidden neurons used to become uncontrollable large when large number of generations is needed.



(a) (b)

Figure 3.13: (a) Classification accuracy on diabetes testing data set (b) Number of neurons used by the network – (NN-GP)

The time taken to train the network that gives the best generalization accuracy uses the shortest time (417.5s). Longer training time does not correspond to higher generalization accuracy. It has been noted that the time taken for NN-SAGP to evolve the NN are 232.7s and 550.6s when using P_m = 0.15 and 0.3, respectively, while the average number of generations are 58.4 and 66.3 for P_m = 0.15 and 0.3, respectively.

Table 3.4: Accuracy, time taken, number of generations and evaluations - diabetes

Diabetes	Probability of growth			
	0.2	0.4	0.6	0.8
Accuracy (%)	75.58	72.53	72.34	74.55
Time (s)	417.5	652.8	630.8	1764.9
Generations	74.0	68.1	64.3	76.9
Evaluations	12950	11917	11253	13458

3.5.2.3 Comparison

NN-GP and NN-SAGP have outperformed all other algorithms in terms of generalization accuracy. Compared to SPANN with a classification accuracy of 70.7%, NN-GP has a notable 17% ((29.3-24.4)/29.3 x 100%) lower percentage error,

while NN-SAGP (P_m = 0.3) attained 18% lower percentage error. Comparing the number of hidden neurons required, NN-SAGP uses the least number of hidden neurons, i.e., 1.4 compared to other algorithms. MPANN and SPANN use about 4.7 and 5.1 times more number of hidden neurons. Using larger number of neurons improved accuracy for both EANN and BPNN; however, this improvement increases too much complexity. The standard deviations of NN-SAGP are also much smaller than MPANN and SPANN. Once again the performance of both algorithms proposed in this chapter is very competitive to each other.

Table 3.5: Comparison of results - diabetes

Algorithm	Test accuracy (%)	No. of hidden neurons
NN-GP(P_g = 0.2)	75.58 ± 2.7584	2.60 ± 1.5776
NN-SAGP(P_m = 0.15)	75 ± 3.6509	1.80 ± 1.3166
NN-SAGP(P_m = 0.3)	75.84 ± 2.5757	1.40 ± 0.6992
MPANN	74.9 ± 6.2	6.6 ± 1.505
SPANN	70.7 ± 5.00	7.166 ± 2.208
EANN1	73.57 ± 2.687	2
EANN2	74.16 ± 2.026	7
BPNN1	73.90 ± 2.072	2
BPNN2	74.42 ± 2.355	7

The *P*-values of the paired *t*-test on diabetes data set show that using P_g = 0.2 outperform P_g = 0.4 and P_g = 0.6 with confidence.

Table 3.6: *P*-values of the paired *t*-test - diabetes

NN-GP				NN-SAGP		EANN		BPNN	
0.2	0.4	0.6	0.8	0.15	0.3	2 HN	7 HN	2 HN	7 HN
-	0.0367	0.0433	0.2669	0.2890	0.4184	0.0459	0.0654	0.1070	0.1677
0.4184	0.0094	0.0169	0.1724	0.2671	-	0.0670	0.0506	0.0569	0.0829

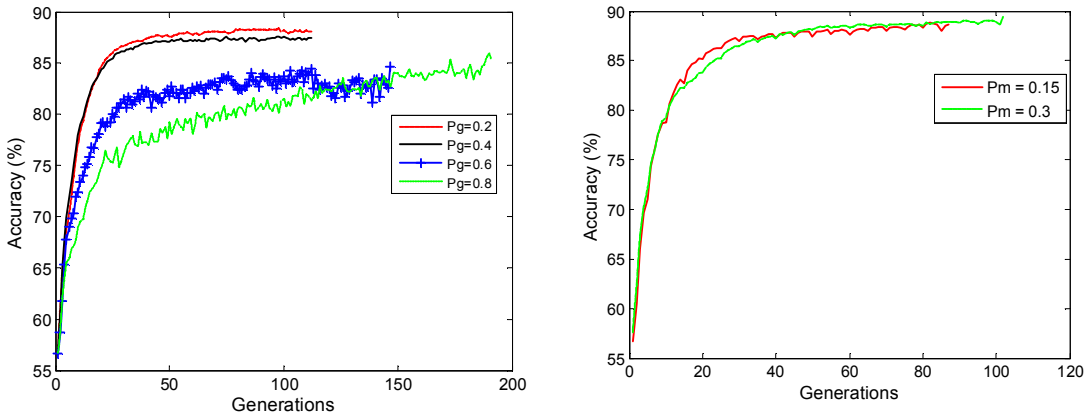
3.5.3 Card Problem

The credit card application problem is a larger data set in terms of larger number of input attributes compared to the previous two problems. This data set serves as an additional indicator to the performance of NN-GP and NN-SAGP. The results on the training and testing data sets are presented in the following sub-sections.

3.5.3.1 Results on Training Data Set

NN-GP: A random initialization of the weights for NN with one hidden neuron obtained an accuracy of about 55% on the training data set. As evolution takes place, classification accuracy increases. Lower P_g obtained better classification accuracy compared to higher P_g. Convergence rate is also faster for lower P_g compared to higher P_g. Higher P_g causes large variation of network sizes in the population, leading to greater disparity in the performance of the different networks and larger diversity in the population. The performance spread of the networks is wider rather than crowding at certain areas of the solution space.

NN-SAGP: NN-SAGP steadily increases the average classification accuracy of the parent population on the training data set. The growth probability as shown in Figure 3.15 decreases over the generations (as the P_g shown is averaged over the runs, and different run uses different number of generations, the sharp increase at the end is due to the P_g for the run with the longest generation).



(a) (b)

Figure 3.14: Classification accuracy on card training data set using (a) NN-GP (b) NN-SAGP

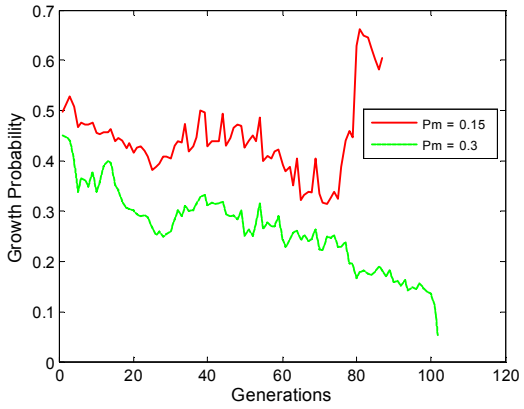


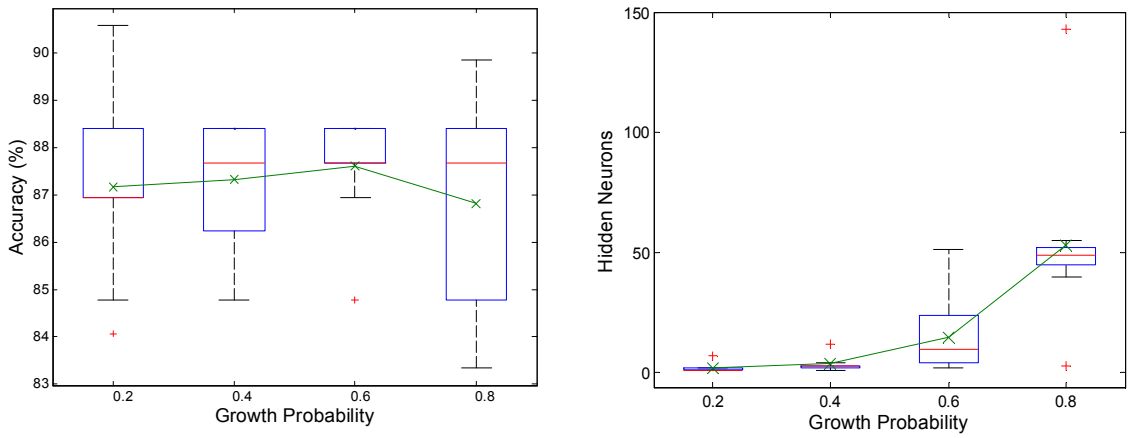
Figure 3.15: Value of growth probability over the generations for card training data set (NN-SAGP)

3.5.3.2 Different Values of Growth Probability on Testing Data Set

The best mean accuracy is achieved when $P_g = 0.6$ is used while the worse mean accuracy is caused by using $P_g = 0.8$. From the box plots, it can be seen that the distribution of the results for $P_g = 0.6$ is more concentrated compared to the others.

The corresponding number of hidden neurons required to obtain the generalization accuracy shown in Figure 3.16(a) is presented in Figure 3.16(b). To obtain the best generalization accuracy, NN-GP uses an average number of 11 neurons. This is three

times more than the number of neurons when using P_g = 0.4, however, without any significant increase in accuracy.



(a) (b)

Figure 3.16: (a) Classification accuracy on card testing data set (b) Number of neurons used by the network – (NN-GP)

The average time taken to train the NN using P_g = 0.4 is 1164s, which is much faster than the time taken to train larger networks that do not guarantee better generalization accuracy.

It has been noted that the time taken for NN-SAGP to evolve the NN are 451.27s and 535.14s when using P_m = 0.15 and 0.3, respectively, which are faster than NN-GP, while the average number of generations are 70.4 and 81.5 for P_m = 0.15 and 0.3, respectively.

Table 3.7: Accuracy, time taken, number of generations and evaluations - card

Card	Probability of growth			
	0.2	0.4	0.6	0.8
Accuracy (%)	87.17	87.32	87.61	86.81
Time (s)	998	1164	3693	14434
Generations	80.6	84	105.8	162.2
Evaluations	14105	14700	18515	28385

3.5.3.3 Comparison

In terms of generalization accuracy, the results obtained by NN-GP and NN-SAGP are better than the results reported for all other algorithms and the number of hidden neurons used by the evolved networks is much smaller. SPANN utilized 70% more hidden neurons compared to NN-GP and about 5.5 times more than what is used in NN-SAGP. The proposed methods in this chapter produce networks with lower complexity yet better generalization accuracy. In addition, judging from the small standard deviation of the accuracy, both the algorithms are indeed the more reliable algorithms to be used for the card data set. The results produced by BPNN and EANN are not able to attain the kind of performance of the proposed algorithms. NN-SAGP (P_m = 0.15) has slightly better performance than NN-GP for this problem. It obtains higher classification accuracy by using a less complex network structure.

Table 3.8: Comparison of results - card

Algorithm	Test accuracy (%)	No. of hidden neurons
NN-GP(P_g = 0.4)	87.32 ± 1.2895	3.50 ± 3.100
NN-SAGP(P_m = 0.15)	87.54 ± 1.7684	1.10 ± 0.3162
NN-SAGP(P_m = 0.3)	86.81 ± 1.3142	1.50 ± 1.2693
MPANN	86.4 ± 4.5	5.00 ± 1.943
SPANN	86.9 ± 4.6	6.0 ± 1.825
EANN1	85.07 ± 1.241	2
EANN2	85.65 ± 1.269	7
BPNN1	84.86 ± 2.199	2
BPNN2	84.28 ± 1.841	7

The results in Table 3.9 might suggest that evolution that uses other P_g apart from 0.4 is able to obtain similar good performance. In addition, the *P*-values against

EANN and BPNN show that both NN-GP and NN-SAGP are able to outperform EANN and BPNN with confidence.

Table 3.9: *P*-values of the paired *t*-test - card

NN-GP				NN-SAGP		EANN		BPNN	
0.2	0.4	0.6	0.8	0.15	0.3	2 HN	7 HN	2 HN	7 HN
0.3968	-	0.3212	0.2874	0.3023	0.2031	8.74×10^{-4}	0.0109	0.0089	0.0023
0.3284	0.3023	0.4648	0.2451	-	0.1222	0.0018	0.0132	0.0087	0.0041

3.6 Conclusions

Training NN using growth probability-based evolutionary technique (NN-GP) has been proposed as a new algorithm to evolve the near optimal number of hidden neurons and weights required by NN for good classification accuracy. In addition, the self-adaptive version of NN-GP is applied to automate the process of finding a suitable P_g through the generations. The performance of the algorithms is enhanced by growing the NN at different rates based on a Gaussian distribution thus avoiding being trapped in local optima.

The algorithms are tested with real-world problems and results from experiments show that NN-GP and NN-SAGP are able to evolve networks with high classification accuracy and low complexity for all problems. The performance of both algorithms are comparable to each other, however the self-adaptive version has the advantage of not requiring the value of growth probability to be determined beforehand. An interesting finding from the experimental results showed that although chromosomes are grown at every generation, it is the growth probability rather than the generation number that has a greater influence on the number of hidden neurons. This prevents

the number of hidden neurons to grow too large when a large number of generations is used.

Chapter 4

An Evolutionary Memetic Algorithm for Rule Extraction

One of the strengths of evolutionary algorithms lies in its inherent global search capability which is able to identify several good regions of the solution space simultaneously. However, when regions of the solution space are being identified, the algorithm continues searching in a stochastic manner while relying on Darwin's survival of the fittest strategy to guide the process. No schemes are used to satisfactorily exploit those identified regions and explore the local landscape thoroughly. In order to overcome this lack of exploitation, a Local Search (LS) technique is often incorporated within EA to complement its search ability. This hybridized scheme is termed memetic algorithms. Memetic algorithms have proved to be successful in several applications in the literature [87][111][118][124][163][187].

With these in mind, this chapter proposes an Evolutionary Memetic Algorithm (EMA) for rule extraction. The main EA evolves the architecture of the rule set while the LS is applied at every generation to fine-tune the rule parameters. This scheme is made achievable by the use of variable length chromosome representation which gives flexibility in the representation of the rule sets and allows easy manipulations by the advanced variation operators, i.e., structural mutation and structural crossover [153]. Two local improvement search algorithms are used in this chapter. The first scheme uses a Micro-Genetic Algorithm (μGA). This simple method provides efficient and guided improvements over the generations. The second scheme

incorporates some ideas from Artificial Immune Systems (AIS) [6][38][83][123][154][165][166]. AIS framework have been applied in several tasks in literature. [39] shows the immune principles can be used to solve complex engineering tasks. Several simplifications of the clonal selection principles are made and CLONAG is given as an algorithm to solve pattern recognition task. In addition, modifications to CLONAG are also given to show how it can be used for optimization problems. AIS for data analysis could be seen in unsupervised machine learning algorithms [167][168]. Watkins and Boggress built on further and developed Artificial Immune Recognition Systems (AIRS) [179] as a supervised classifier and these systems follow a *k*-nearest neighbor scheme. AIRS as a classification tool has shown to be very competitive to other existing algorithms in the literature [54][60][180].

The major evaluation metric for classification rule sets in the literature has been classification accuracy. However, there are other aspects which are equally important when considering the performance of rule sets. Apart from how accurate the rule sets are able to classify a data set, the coverage on the data set is also important. The ideas of support and confidence level borrowed from association rule mining [77][139][182] are incorporated into the fitness evaluation function. The effects of these two factors are analyzed and discussed.

In Section 4.1, a brief introduction of AIS is given. The details of the proposed features and operators of EMA are described in Section 4.2 while the algorithm overview is given in Section 4.3. Section 4.4 presents the local search algorithms. Experimental setup and data sets used are described in Section 4.5. Results from experiments on real-world benchmarking data sets are given and analyzed in Section 4.6. Conclusions are subsequently drawn in Section 4.7.

4.1 Artificial Immune Systems

Artificial immune systems [38][102] are based on the working mechanisms of the human immune system. Basically, the human immune system is built upon an innate immune system and an adaptive immune system [154]. The innate immune system does not target any specific antigenic stimulus but is activated upon encountering a general stimulus. On the other hand, the adaptive immune system is targeted at specific antigenic stimulus with cells that fit the antigens producing antibodies to eliminate them.

The clonal selection principle [34][38][39][54][154] is used to describe the way the adaptive immune system works. It is interested in modeling the adaptive immune system as it has a memory capability to store information after its primary encounter. Hence, it is able to response faster and more efficiently for the next encounter, this provides immunity for the body against the same infection.

Basically, the adaptive immune system consists of the lymphocytes which are made up of the B-cells and the T-cells. The T-cells are known as the accessory cells, secreting lymphokines which act as stimulus causing the B-cells to proliferate. The B-cells are the antibodies producing cells. When the B-cells receive stimulus from the T-cells, the B-cells having its receptors that fit to the antigens will be selected to proliferate by a cloning process where replica of the cells are made. These cloned cells will differentiate and mature into plasma cells or terminal antibody secreting cells. These cells are non-dividing and will produce the antibodies to combat the antigens. In addition, a portion of the B-cells will become long lived B-memory-cells. These cells will multiply quickly during secondary response upon the same antigenic stimulus encountered in the primary response to produce antibodies with high affinity.

4.2 Algorithm Features and Operators

The various features and operators of the algorithm are given in the following sections.

4.2.1 Variable Length Chromosome

In this chapter, Pittsburgh encoding is applied and the encoding of a rule set would be much longer than that of the Michigan approach. With more parameters within a chromosome, the search space is inevitably enlarged and the time taken for the Pittsburgh approach to find a good solution is also lengthened. Appropriate representation and specific variation operators are required to ease the implementation.

The rule set chromosome structure representation often used in the literature is the fixed length structure representation. Fixed length representation is not suitable for design problems where various parameters are to be concurrently evolved as the use of it might cause invalid solutions after the variation operators are applied, and the representation itself inherently presents several constraints and limitations.

This chapter proposes the use of variable length chromosome representation [153] to represent the rule set topology. This representation is efficient and simple for the application of the variation operators and it provides the additional flexibility required for the concurrent evolution of the various parameters like the number of rules within a rule set, the boundary conditions for each feature, the masking string, the operator string and the predicted outcome. Each chromosome within the population represents a rule set which is made up of different number of rules. Larger number of rules within a rule set means higher complexity. Each rule within the rule set is seen as a block unit which consists of four allele strings (details of the allele strings are given in

the following sub-sections), this provides an efficient manner for deletion or addition of the rules.

4.2.1.1 Boundary String

The first string encodes the boundary values for each input feature of the data set. The lower boundary for the *i*th input feature is denoted by L_i while the upper boundary is denoted by U_i. This boundary value gives the numeric threshold for the input feature and is to be used concurrently with the operator string. The length of this string and also the masking and operator strings depend on the number of input features of the given problem.

4.2.1.2 Masking String

The masking string has the effect of feature selection as not all the features presented in the data set are necessary for good classification results. Though several features are present in a data set, very often only a fraction of these features are useful for good classification accuracy [30]. The inclusion of all the features may even deteriorate the classification performance as they may mislead or interfere with the good features. The masking allele is a binary bit string indicating whether a given input feature should appear in the rule. A bit 0 at the *i*th position indicates exclusion of feature *i*, and the corresponding operator and boundary conditions would be excluded too. Similarly, a bit 1 indicates inclusion of the feature.

4.2.1.3 Operator String

The operator string indicates the inequality and equality operators to be applied on the boundary string. The four operators encoded are namely, greater than or equal

"≥", less than or equal "≤", within a range of "≤ ≤" and less than or greater than "≤ ≥", given by $x \geq L$, $x \leq U$, $L \leq x \leq U$ and ($x \leq L$ or $x \geq U$), respectively, where x is a given numeric attribute.

4.2.1.4 Class String

If any of the rules is able to cover or capture the instance, the instance would be classified as the corresponding class else it would be classified as the general class. There are several methods used to determine the general class. Some methods for example are the majority class given in the training data set, random assignment, evolved using EA, etc. [160][161][162].

Figure 4.1 illustrates a variable length chromosome representation of a rule set with m number of rules for a data set with n number of input features.

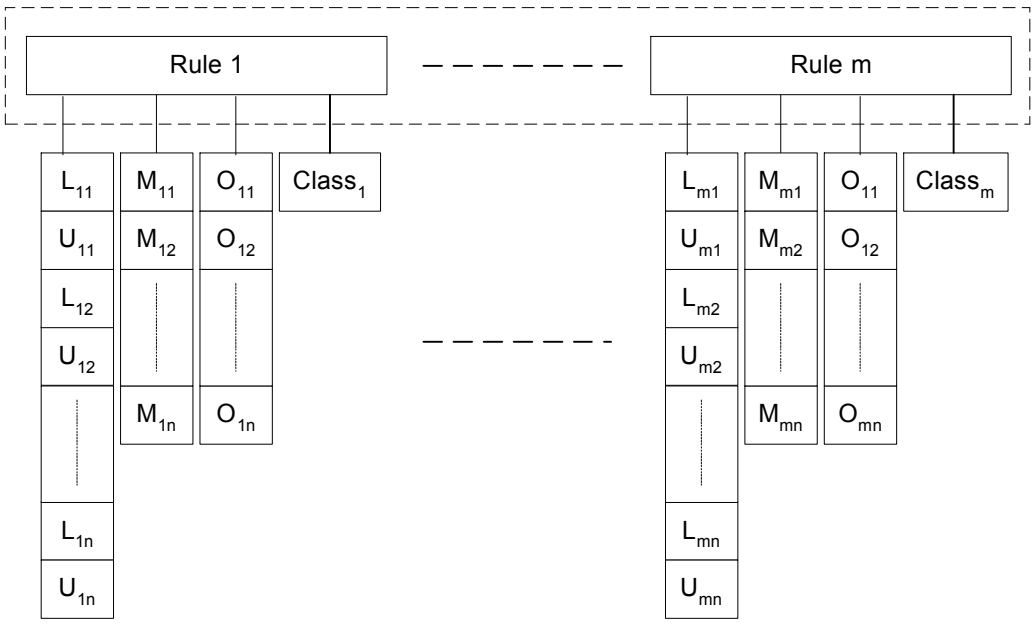


Figure 4.1: Variable length chromosome representation of a rule set

4.2.2 Fitness Evaluation

Several different evaluation metrics for rule evaluation are presented in the literature and the most commonly used metric is the classification accuracy, which has a more direct relation to the generalization accuracy. Sensitivity and specificity [119][134] which are statistical tests for binary classification are also often used; however, these are only applicable for Michigan rules. The less commonly considered metrics are comprehensibility and interestingness. The comprehensibility metric is a subjective measure on how clear and easy a rule is interpretable by humans. Generally, rules that are incomprehensible to humans are often useless in data mining or knowledge discovery because such rules are not beneficial to the users. Interestingness is a measure of how common a rule is. A common rule is deemed less interesting than an uncommon rule.

In this chapter, the fitness function used to evaluate the rule sets comprises of the classification accuracy being the major metric and in addition, ideas borrowed from the evaluation of the association rules [4] are also used. Association rules are rules that attempt to find interesting relationships among variables in a database. These rules are often used for databases that would present some trends within the variables, e.g., supermarket database, earthquake database, etc. In a supermarket database, an association rule may look like, <if a customer buys flour and egg, the customer would buy sugar too>. The common metrics for association rules are the support and confidence level [139]. Support of *A or* sup*(A)* is a measure of the number of transactions that contain the item set within the database whereas confidence of an association rule is:

$$\text{confidence}(R) = \frac{\sup(A \bigcup C)}{\sup(A)} \tag{4.1}$$

where $\sup(A \bigcup C)$ is the item set containing A(antecedent) as well as C (consequent).

In the context of this chapter, instead of using support and confidence level for a rule, it is applied on a Pittsburgh rule set. The support would then be modified as measuring the coverage of the entire rule set, which is the ratio of the number of instances covered by the rule set to the total number of instances in the data set:

$$\sup(RS_i) = \frac{B_i}{N} \bullet 100\% \tag{4.2}$$

RS_i is the ith rule set, $i \forall [1,...M]$ where M is the number of rule sets to be evaluated. B_i is the number of instances covered by the rule set and N is the total number of instances.

The confidence factor will measure how accurate a rule set is on those instances that are covered.

$$\text{confidence}(RS_i) = \frac{(B_i \bigcup CR_i)}{B_i} \bullet 100\% \tag{4.3}$$

where $(B_i \bigcup CR_i)$ is the number of instances covered and correctly classified by the rule set.

The support factor and confidence factor will provide additional indicators for the performance of the rule set.

Classification accuracy is the major component used in the fitness function as eventually it is the generalization accuracy which is of significant importance. This classification accuracy measure is given as:

$$\mathrm{CA}(RS_i) = \frac{TC_i}{N} \bullet 100\% \tag{4.4}$$

TC_i is the total number of correctly classified instances and N is the total number of instances in the training data set, i.e., $(TC_i \leq N)$.

In order to integrate the three components as a fitness function, a weighted approach is used. Depending on the user's priority, the weights are set accordingly. The Modified Weighted Fitness Function (MWFF) is given in Equation 4.5. In this chapter, the classification accuracy is considered the most important component. w_1 is set as 1 while $w_2 = w_3 = 0.2$.

$$\mathrm{MWFF} = w_1 \bullet \mathrm{CA} + w_2 \bullet \mathrm{Support} + w_3 \bullet \mathrm{Confidence} \tag{4.5}$$

4.2.3 Tournament Selection

Offspring are created using structural crossover of the parents. Parents with higher fitness are selected for crossover process and this is done using binary tournament selection with replacement. In binary tournament selection, two parents are randomly selected and the parent with higher fitness would be taken as a snapshot for offspring creation process. The two parent chromosomes are then replaced back into the parent population pool. The selection of the second parent for crossover follows the same procedure.

4.2.4 Structural Crossover

Crossover process is required so that useful information is exchanged between parents and passed to the offspring. Since the standard chromosome representation is

not used, application of standard genetic operators might not be suitable. A problem specific crossover operator is applied.

Firstly, crossover points are only allowed at the junctions between each rule, hence crossover is carried out in terms of rule blocks. The crossover point should not occur in between each rule which would break up the rule's strings. Secondly, the crossover process is done in a shuffled manner. This shuffled crossover process starts with combining the rules of both selected parents into a common pool. The first selected rule will be assigned to the first child while the second selected rule to the second child. The remaining rules are then distributed randomly between the two children. Since rules are considered sequentially within a rule set, rules at the bottom might not have a chance to be considered even though they are good rules as those rules on top would have already classified the instance. One major advantage and necessity of doing structural shuffling crossover is to bring those rules that are at the bottom to the top of the rule set, therefore having the opportunity of classifying the instances. Figure 4.2 shows an example of the crossover process. It can be seen that after the crossover process, the rules are randomly ordered and the number of rules of a rule set may vary too.

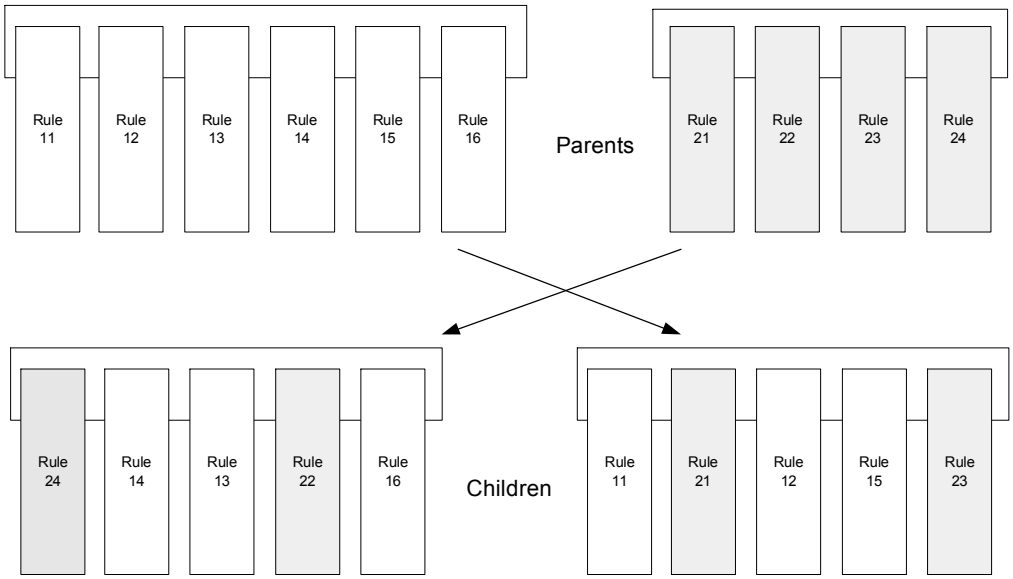


Figure 4.2: Structural crossover process

4.2.5 Structural Mutation

The mutation operator provides a mechanism for solutions to escape from local regions and to increase diversity. The encoding of Pittsburgh rule sets results in higher complexity than in Michigan approach, hence dedicated variation operators are needed when applying crossover and mutation. The structural mutation is applied based on the probability of structural mutation, P_{sm}. Structural mutation involves the addition or deletion of rules. With this insertion and deletion, the resulting rule set could translate into a very different rule set, ensuring that new areas of the search space are constantly being explored. Figure 4.3a shows an example of rule addition while Figure 4.3b shows an example of rule deletion. The positions of insertion and deletion of the rules are randomly chosen.

The number of rules to be added or deleted is based on a Gaussian distribution [7]. Using the Gaussian distribution has the advantage of allowing more flexibility on the number of rules to be added or deleted. Figure 4.4 shows the decisions based on the Gaussian distribution.

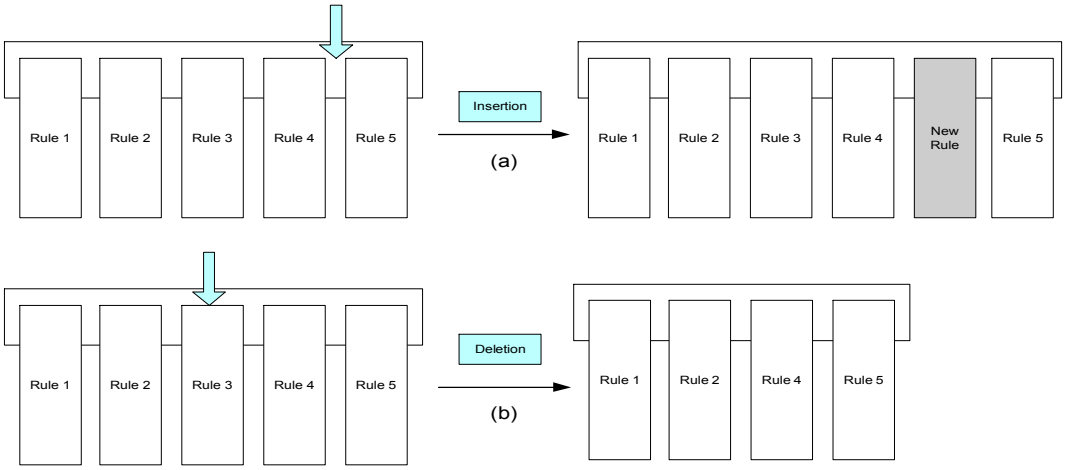


Figure 4.3: Structural mutation – (a) Addition (b) Deletion

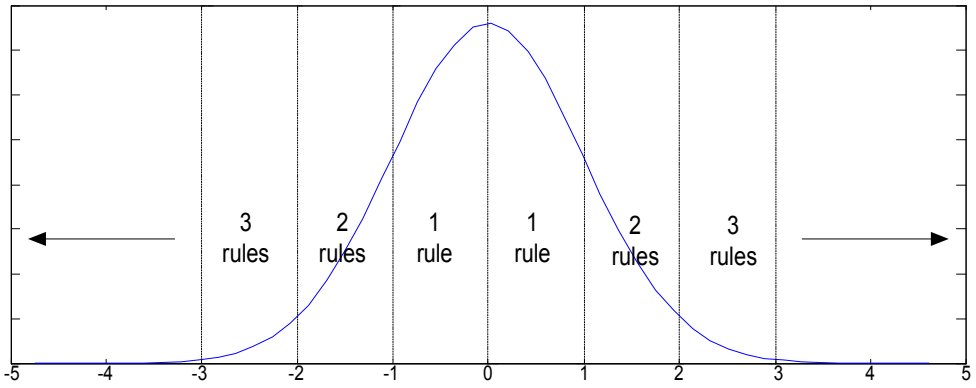


Figure 4.4: Gaussian distribution for structural mutation

4.2.6 Probability of Structural Mutation

The probability of mutation plays an important role in the mutation process. If the mutation probability is small, there would be too many similar chromosomes. On the other hand, having a large mutation probability directs the search towards random search and increases the possibility of disrupting the chromosomes that may carry good solutions. Researchers have acknowledged the importance of this parameter and many works have been done. Generally, research papers have showed that using a fixed mutation probability throughout the whole evolution process is not optimal and

efficient, and several works are done using varying mutation probability [49][144][155][164]. [49] investigated the use of varying mutation probability and showed its effectiveness over fixed mutation probability. Similarly in [155], the algorithm uses a mutation probability with high value during the initial generations and with a drastic drop in value for the later generations. This translates to exploration in the initial phase and exploitation in the later phase. This mutation scheme is different from the other adaptive mutation schemes where the mutation probability decreases proportionally or gradually as the generation increases.

In this chapter, the probability of structural mutation P_{sm} changes based on the scheme of [155], however it is modified to suit the problem discussed. The equation used in this chapter is given in Equation 4.6.

$$P_{sm} = \begin{cases} 0.7\left[1-\left(\dfrac{n}{genNum}\right)^2\right], & 0 \le n \le 0.2 \bullet genNum, \\ 0.2\left(\dfrac{n-genNum}{genNum}\right)^2 + 0.05, & 0.2 \bullet genNum < n \le genNum, \end{cases} \tag{4.6}$$

where n is the current generation of the evolution process, $genNum$ is the maximum generation number.

Figure 4.5 shows the values of the structural mutation probability as the evolution proceeds through the generations. This ensures that during the initial stage of evolution, there are large jumps in the search space while at the later stage, it narrows down to searching within the neighborhood.

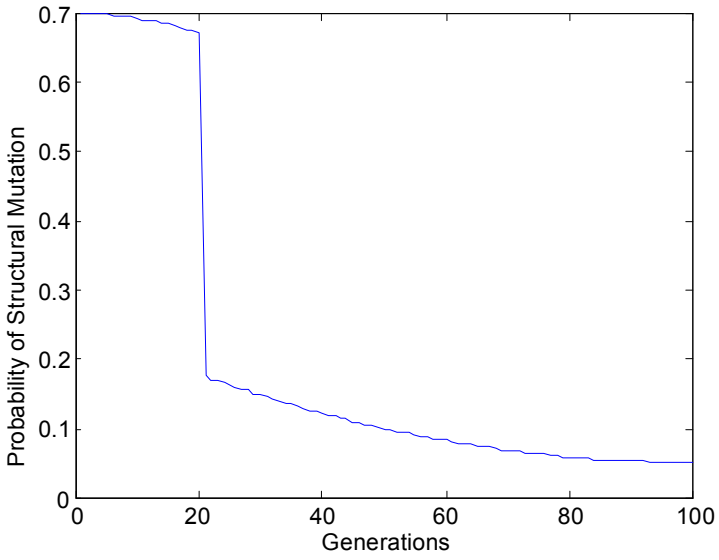


Figure 4.5: Mutation scheme

4.2.7 General class

The general class used in this chapter is the major class of the training data set. However, from the experimental results provided in the later sections of this chapter, what is used as the general class is not an issue as the support for the rule sets evolved is able to achieve almost 100% coverage for both the training and testing data sets.

4.2.8 Elitism and Archiving

Elitism is required so that good solutions would not be lost and could be propagated through the generations. In addition, elitism has the effect of improving convergence speed.

The archive acts as candidates for the parents in the next generation. The chromosomes in the archive go through local search with the tuned chromosomes emerging as the parents for the next generation. The size of the archive is set as the size of the parent population.

4.3 Evolutionary Memetic Algorithm Overview

This section presents the synchronization of all the different variation operators for the overall rule extraction algorithm.

4.3.1 Training Phase Overview

The objective of the training phase is to have an algorithm that is capable of optimizing the architecture of rule sets and parameters of the rules concurrently. A few features, such as variable length chromosome representation, specialized genetic operators in the form of structural mutation and crossover, and local exploitation scheme are incorporated so as to achieve the objective. The rules evolved should be high in support and confidence level, and most importantly produce good classification accuracy.

Figure 4.6 shows the flowchart of EMA for rule extraction. The algorithm first initializes a population of rule sets and evaluates their fitness. These rule sets act as the initial parent population for offspring creation and the population goes through a series of binary tournament selection and structural crossover based on P_{sc} (structural crossover probability) to exchange good information to pass on to the children. These newly created offspring would undergo structural mutation based on P_{sm} (structural mutation probability) in order to vary the number of rules in a rule set. The archive size selected is the same as the parent size (β), i.e., the fittest β chromosomes in the children population are archived. These archived individuals would undergo either of the two local search operators presented in this chapter, i.e., the μGA or the AIS inspired local search. The fitter individuals emerging from the local search would be brought forward to the next generation as parent chromosomes. This process

continues until the overall stopping criterion, which is the maximum number of generations allowed, is met.

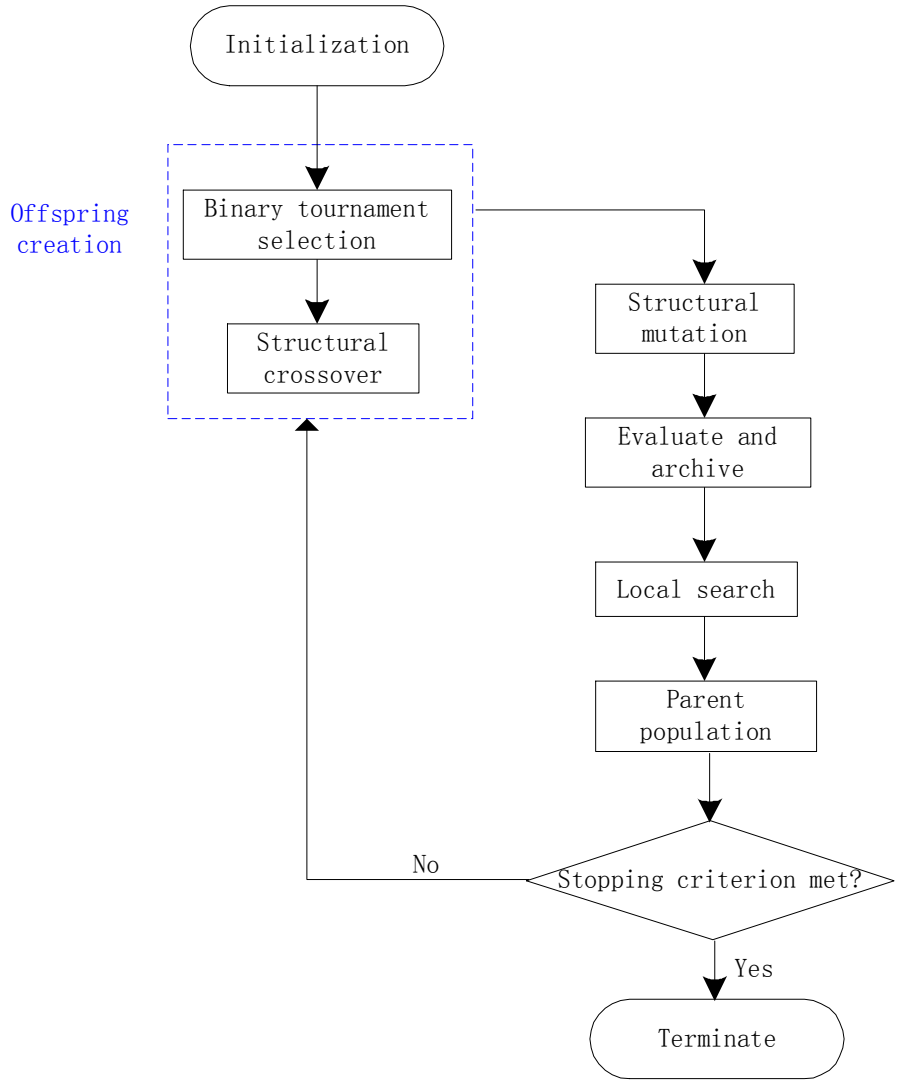


Figure 4.6: EMA overview

4.3.2 Testing Phase

The rule set that has the highest fitness on the training data set would be applied on the testing data set. Each instance in the testing data set would be presented to the rule set for classification. If the first rule does not capture the instance, this instance will be considered by the subsequent rules. If none of the rules is able to classify this

instance, the class will be stated as the general class. It is important that rule sets have high support level during the training phase, as high support level on the training data set would most probably translate to higher coverage on the testing data set.

4.4 Local Search Algorithms

LS algorithms are used to complement the global search capability of the EA by fine-tuning the parameters of the rules. This section introduces two variants of LS algorithms that are incorporated into the main rule extraction algorithm. The main differences between both the LS algorithms lie firstly, in the AIS inspired LS, cell replicates quickly by cloning to form a large pool of cells which are the same as the original cell. In μGA the offspring created are different from the parents and among themselves. Secondly, both crossover and mutation are used in μGA while only mutation is used in AIS inspired LS. Thirdly, the basic conceptual frameworks of both the algorithms are different.

As the AIS inspired LS here does not aim at emulating any specific AIS algorithms but rather to use some of the basic principles to create another version of the LS for comparison, the AIS inspired LS here would not be too different from the μGA LS apart from those points as highlighted.

4.4.1 Micro-Genetic Algorithm Local Search

The framework of the first LS, μGA, is similar to a normal evolution process; however, it differs in its purpose, implementation and operation. The target of the main EA is to optimize the structure of a rule set and exchange good rules among the rule sets through the use of structural mutation and structural crossover. In micro-genetic algorithm local search, the structure of the rule sets is maintained throughout

the evolution process while the parameters of the rules are being exchanged and mutated. In this manner, the parameter values are optimized to fit the rule set structure.

Figure 4.7 shows the overview of the μGA LS. Each individual from the archive of the main algorithm would go through one round of LS process. A parent goes through LS crossover to produce the children population. After the crossover process, all the children undergo LS mutation. These mutated children are evaluated and the fittest child is selected as the parent to be used for crossover process in the next generation. This process is repeated until the overall stopping criterion, which is the maximum number of generations of the LS, is met. When the LS process ends, the fittest child would be returned to the main algorithm to be the parent for the next generation.

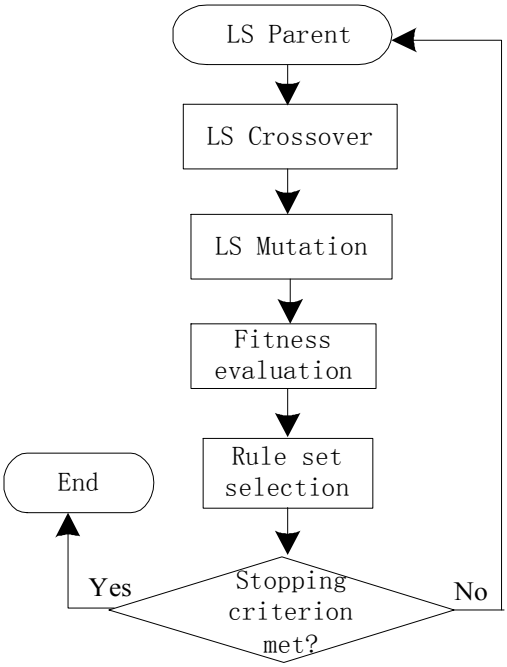


Figure 4.7: Micro-genetic algorithm local search overview

4.4.1.1 Local Search Crossover

The LS crossover is done among the rules of a rule set as depicted in Figure 4.8. Using one rule set, it is able to create multiple different children by choosing different rules and different crossover points. In exceptional cases, where the chosen rule set contains only one rule, no crossover can be done and it is replaced by the AIS inspired LS. If the chosen rule set contains two rules, different children are created by choosing random points for crossover among the two parent rules. The allowed crossover points of the boundary strings are at the intersection of each attribute to prevent separating the lower and upper boundaries of an attribute. This constraint is applied to avoid any infeasible solution after the crossover.

For rule sets with three or more rules, two random rules and one crossover point are chosen to create a child. This process is repeated to create different children. Once a crossover point is selected, this point is consistent for both the rules. Figure 4.8 shows a rule set containing three rules for a four input feature problem. Rules 1 and 3 are selected for crossover process and the selected crossover point is between feature 1 and feature 2. No change is made to the class field of each rule.

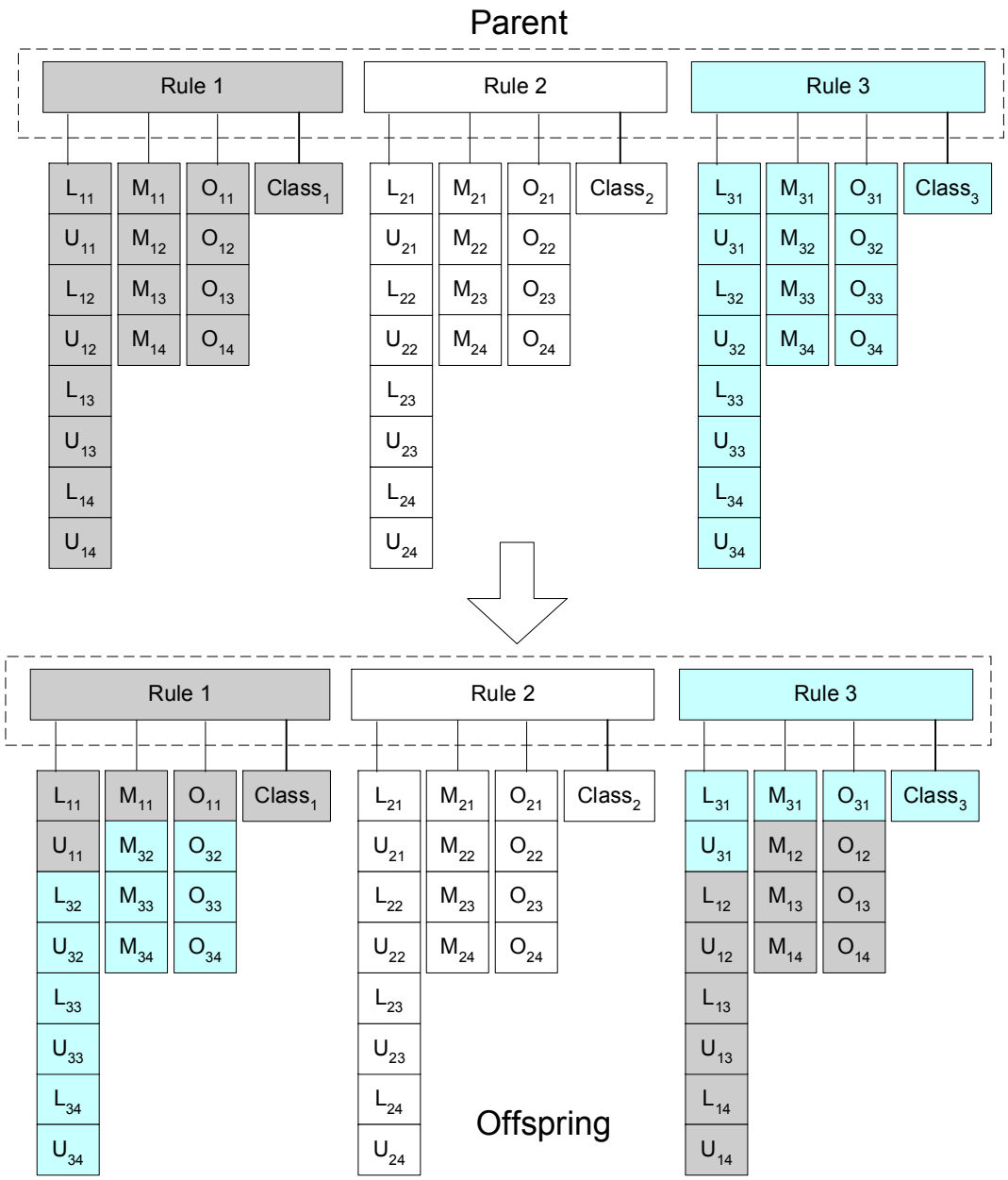


Figure 4.8: Local search crossover process

4.4.1.2 Local Search Mutation

By using the mutation operator, the algorithm is able to explore new areas of the search space and has the possibility of getting a better solution. Each rule consists of four fields, namely the boundary, masking, operator and class. The LS mutation used is field specific and due to the limitations of each field, constraints are also applied accordingly. The probability that an allele will be mutated is based on the LS

mutation probability P_{lsm}. This mutation probability is generation/iteration-based; however, it is different from the structural mutation probability. As the number of generations/iterations used in the local search is smaller than in the main algorithm, the structural mutation probability does not seem suitable. The LS mutation scheme used here is given in Equation 4.7.

$$P_{lsm} = 0.2 \bullet \left(\frac{1}{g}\right) + 0.05 \quad ,\{g \in Z_{+} : [1, gNum]\} \tag{4.7}$$

where g is the current generation/iteration number and $gNum$ is the maximum number of generations/iterations allowed.

The mutation details for each string are given as:

***Boundary String**:* Real-value mutation is used to change the values of the boundary conditions. A random value between [-0.1,0.1] is added to the allele. If the value of the allele exceeds the range [0, 1] after undergoing mutation, boundary conditions given in Equation 4.8 are enforced.

$$v_q = \begin{cases} 1, & \text{if } v_q > 1 \\ 0, & \text{if } v_q < 0 \end{cases} \tag{4.8}$$

where v_q = mutated value of allele occupying the qth position of the string.

***Masking String**:* The mutation of the masking string takes the form of a bit flip operator. A bit 1 is mutated to be bit 0 and vice versa.

***Operator String**:* Mutation on an operator allele would cause that allele to take any of the other three types of operators.

***Class String**:* The number of possible mutations is dependent on the number of output classes in the problem. A random class would be used to replace the existing class in mutation.

4.4.2 Artificial Immune Systems Inspired Local Search

Though AIS are widely used for classification, they are seldom used to provide high level linguistic rules. A number of works, including Carvalho and Freitas [24][25], use immological algorithm to classify examples belonging to small disjuncts. The set of small disjuncts would then be able to cover a large set of examples. In [167], an AIS is used to discover fuzzy classification rules whereas in [26] an antibody represents a set of fuzzy classification rules. In addition, AIS are usually used as main algorithms for classification. The construction of an AIS inspired LS algorithm to fine-tune the rules in a rule set would open up new interesting avenues on the hybridization of EA with AIS.

The LS algorithm incorporates the characteristics of the clonal selection principles which includes hyper-mutation, affinity selection and clonal expansion. The algorithm is a simple one which would not cover all aspects of the clonal selection principles, however it is one that follows its general framework. The analogous of the antibodies of the natural immune systems is the solution to the problem, i.e., the rule sets. The antigen is the optimal solution of the problem in terms of accuracy. The affinity measure is a measure of the Euclidean distance of the antibodies and the antigens in the solution space, i.e., an error measurement. The higher the accuracy of the antibodies the closer it is to the antigen, therefore higher affinity. Proliferation of the B-cells takes place by exact cloning of the antibody that has the highest affinity. Hyper-mutation takes the form of LS mutation. The memory cell of the proliferated B-cells is the best antibody solution from the cloned cell, which is also the antibody selected for cloning.

The LS starts with getting an antibody from the archive of the main algorithm. The archive of the main algorithm stores a selected pool of chromosomes with better

fitness from the children population. This antibody proliferates by cloning based on a clone rate. The cloned cells then go through diversification and maturation through a LS mutation process. Each of the cloned cell is evaluated based on its affinity with the antigen, which in this case is the optimal solution. The cell with the highest affinity (lowest error) is selected as the antibody or as the memory cell for secondary response. This process is repeated over several times. The flow chart of the LS which is inspired by the AIS is given in Figure 4.9.

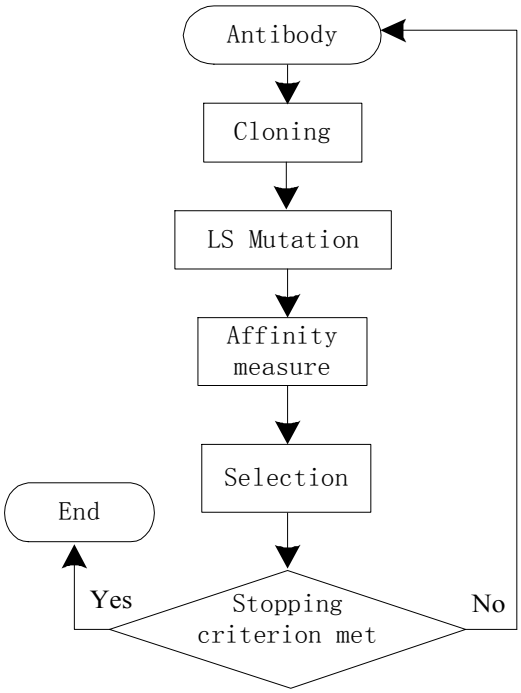


Figure 4.9: AIS inspired local search algorithm

4.5 Experimental Setup and Data Sets

This section states the experimental setup and data sets used. The parameter settings used are also justified in this section.

4.5.1 Experimental Setup

EMA is implemented using the MATLAB technical computing platform and the corresponding experiments are performed on Intel Pentium 4 2.8 GHz computers. Twenty independent runs are performed for each data set using the corresponding experimental setup.

As different data sets have different characteristics with different convergence rates and requirements, specific settings that suit each data set are required. Whether a sample data can be easily classified depends on several factors both in the input space and the output space. Generally, problems with larger input and output dimensions have higher complexity, and problems that require more linearly separable hyper-planes to correctly classify the inputs are also more difficult problems. These aforementioned issues imply that there is no one for all optimal settings to be used for the algorithms to be applied on all the data sets.

The parameter settings used for the main algorithm and the LS are tabulated in Table 4.1 and Table 4.2, respectively. As shown in Table 4.1, the settings for all the data sets are generally consistent apart from the parent size, offspring size and number of generations. The number of generations used depends on whether the EMA algorithm has converged in the training data set. The diabetes data set requires greater number of generations to converge as compared to the other two data sets (this will be evident later in the results section). In terms of number of evaluations per generation, the diabetes requires 175 (parent: 25 + offspring: 150) evaluations which is also higher than the cancer and iris data sets which requires only 150 evaluations. The structural crossover probability used is 0.7 which is consistently applied across all generations. This probability is required to maintain constant exchange of useful information among the chromosomes. Unlike the structural crossover probability, the

structural mutation probability declines over the generations as small changes is preferred over large changes at the end of the training generations.

Table 4.1: Parameter settings for EMA

	Cancer	Diabetes	Iris
Populations	Parent: 30 Offspring: 120	Parent: 25 Offspring: 150	Parent: 30 Offspring: 120
Archive size	30	25	30
Structural crossover probability	0.7	0.7	0.7
Crossover type	Chromosome	Chromosome	Chromosome
Generations	50	100	50
Structural mutation probability	Generational	Generational	Generational
Mutation type	Structural	Structural	Structural

As LS is performed on each chromosome in the archive, the parent/antibody size in the LS is 1 in each round. LS focus mainly on local fine-tuning, therefore the number of iterations and offspring creation are lesser than the main algorithm. The μGA LS uses a crossover probability of 0.9 among the rules of the rule set whereas the AIS inspired LS uses cloning only. Since the number of generations/iterations used for LS is only 10, the type of generation/iteration-based mutation between the main algorithm and LS is different. The difference between the crossover operators for the main algorithm and μGA LS lies in the former uses chromosome-based, meaning it is done between two chromosomes, while the latter uses rule-based, which is done among the rules in each rule set.

Table 4.2: Parameter settings for local search

	μGA	AIS
Populations	Parent: 1 Offspring: 5	Antibody: 1 Clone rate: 5
LS crossover probability	0.9	No
Crossover type	Rule	No
Generations/Iterations	10	10
Cloning	No	Yes
LS mutation probability	LS generational	LS iterational
Mutation type	LS mutation	LS mutation

4.5.2 Data Sets

Data sets from the medical and botanical fields are being used to validate the proposed algorithms. The cancer, diabetes and iris data sets represent both binary and multi-class classification problems. The selection of these data sets would be able to verify how the algorithms perform on problems with different number of inputs and outputs. For each data set, 75% of the data is used as training set while the remaining 25% is used as testing set. Values of the data sets are normalized in the range of [0,1].

4.6 Experimental Results and Analysis

In this section, the experimental results are presented and discussed. Comparisons are made against rule extraction without local search (No LS) and a well-known rule-based algorithm in the literature, i.e., PART [52]. The parameter settings and operators used for No LS are similar to EMA with the exception that there is no fine-tuning of the parameters by LS. The comparisons are not meant to be exhaustive but rather it provides an indication of how EMA performs in addition to the advantage of being able to generate high level comprehensible linguistic rules.

4.6.1 Training Phase

Figures 4.10 to 4.12 show the fitness level, classification accuracy, support level and confidence level achieve when using EMA-μGA, EMA-AIS and No LS on the cancer, diabetes and iris training data sets.

Cancer Data Set: From Figure 4.10, it can be seen that the fitness from all the algorithms has converged at the end of 50 generations. All the algorithms presented a rather smooth convergence curve rather than a fluctuating and noisy one, meaning the algorithm is able to improve the classification accuracy on the training data set gradually and steadily, with each generation getter fitter. Compared to No LS, both EMA-μGA and EMA-AIS are able to obtain higher fitness values on the training data set, while both of them have very similar performance. Using LS converges quickly after a few generations. The first generation accuracy is also higher for EMA-μGA and EMA-AIS due to the presence of LS.

The highest classification accuracies obtained by EMA-μGA and EMA-AIS are around 97.6% while that of No LS is around 95.7%. Since the support on the data set is about 100%, this classification accuracy performance is very much solely due to the performance of the rule sets evolved rather than dependent on using the general class. In addition, when the support is almost 100%, the confidence level is reflective of the accuracy of the rule sets on the entire data set, and the fitness, accuracy and confidence graphs would look similar with the only difference in the scaling.

The support level for the LS algorithms started off with having around 97% coverage and reaches 100% coverage, after which it remains consistent. Without using LS the support is also able to obtain 100% at the end of the 50 generations. This implies that the incorporation of the support factor into the weighted fitness function

is effective in improving the support level to obtain almost 100% on the training data set.

Diabetes Data Set: Figure 4.11 shows that the algorithms, when applied on the diabetes data set require a larger number of generations for convergence. The convergence rate is also slower than that for the cancer data set. Once again, all the algorithms presented a smooth graph that has fitness getting better as the generation increases. The classification accuracies obtained by both EMA-μGA and EMA-AIS at the end of the generations are about 82% while No LS gave 77.8%. Incorporation of the LS is required for the algorithm to perform well on the training data set. The support on training data set increases sharply to 100% during the initial generations which implies that most of the search effort after the initial phase of the algorithm is concentrated on increasing the confidence and accuracy level.

Iris Data Set: The fitness level for the algorithms with LS are always higher than No LS. The classification accuracies of both EMA-μGA and EMA-AIS rises from 90% to around 98.9% at the end of the generations. No LS algorithm rises from 70% to 94.2%. The performance of No LS in terms of fitness and classification accuracy could not match both the EMA LS algorithms, while both the EMA-μGA and EMA-AIS have comparable performance. This shows that LS is important in improving the results on training data set. The support on the iris data set reaches 100% after the initial phase of training. The observations made from the iris data set shows similar trends with cancer and diabetes data sets.

Generally, different data sets require different number of generations for convergence. In addition, the algorithms are able to improve the support on the

training data set rapidly, showing confidence that the accuracy is solely due to decisions of the rule sets rather than the general class rule.

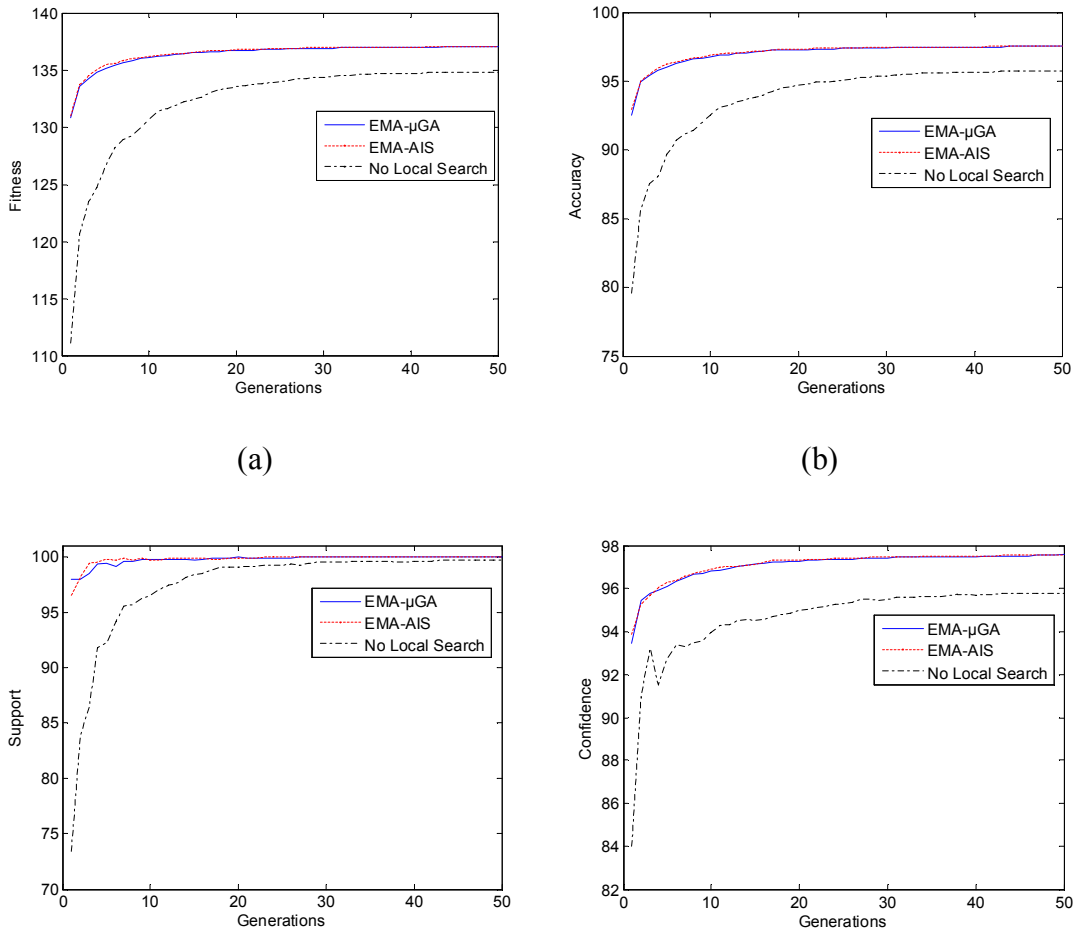


Figure 4.10. Training results for the cancer data set

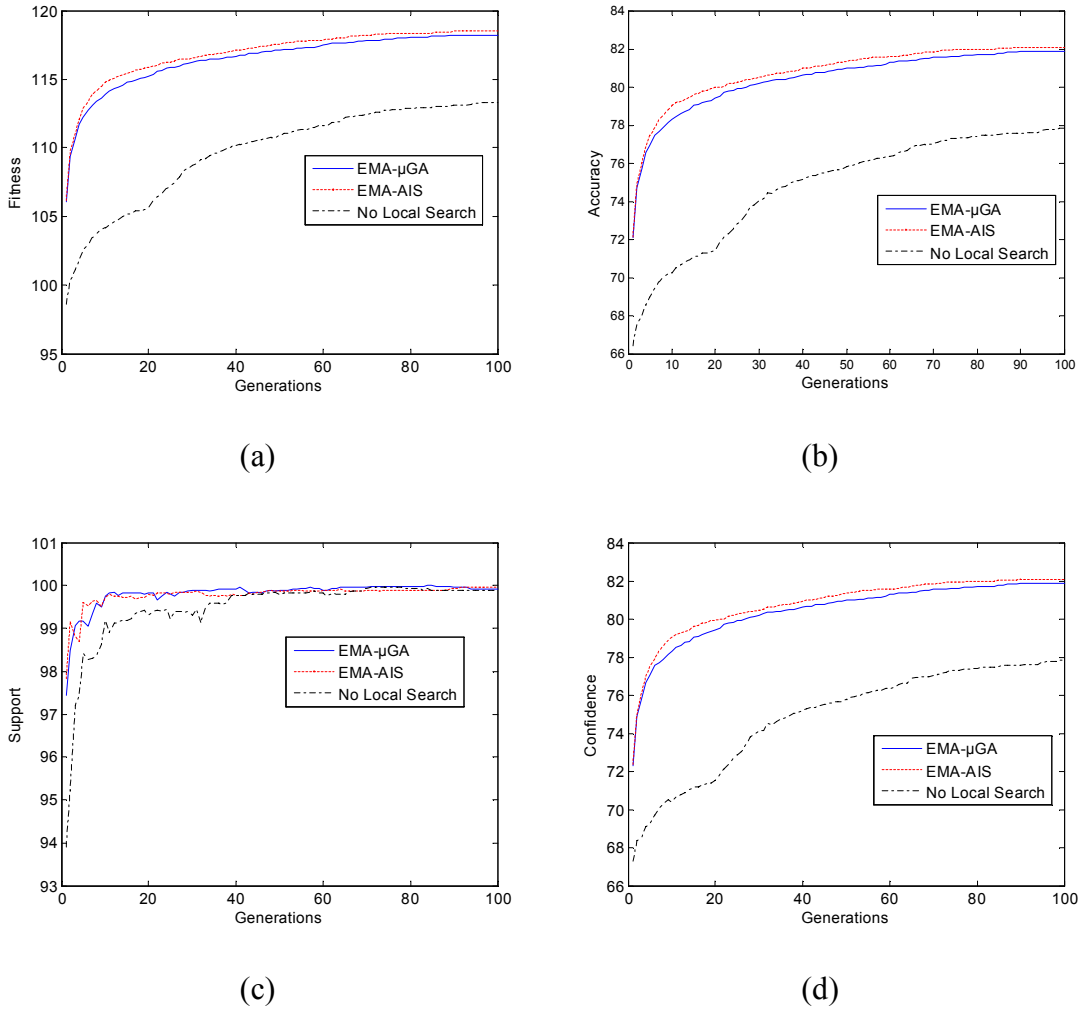


Figure 4.11. Training results for the diabetes data set

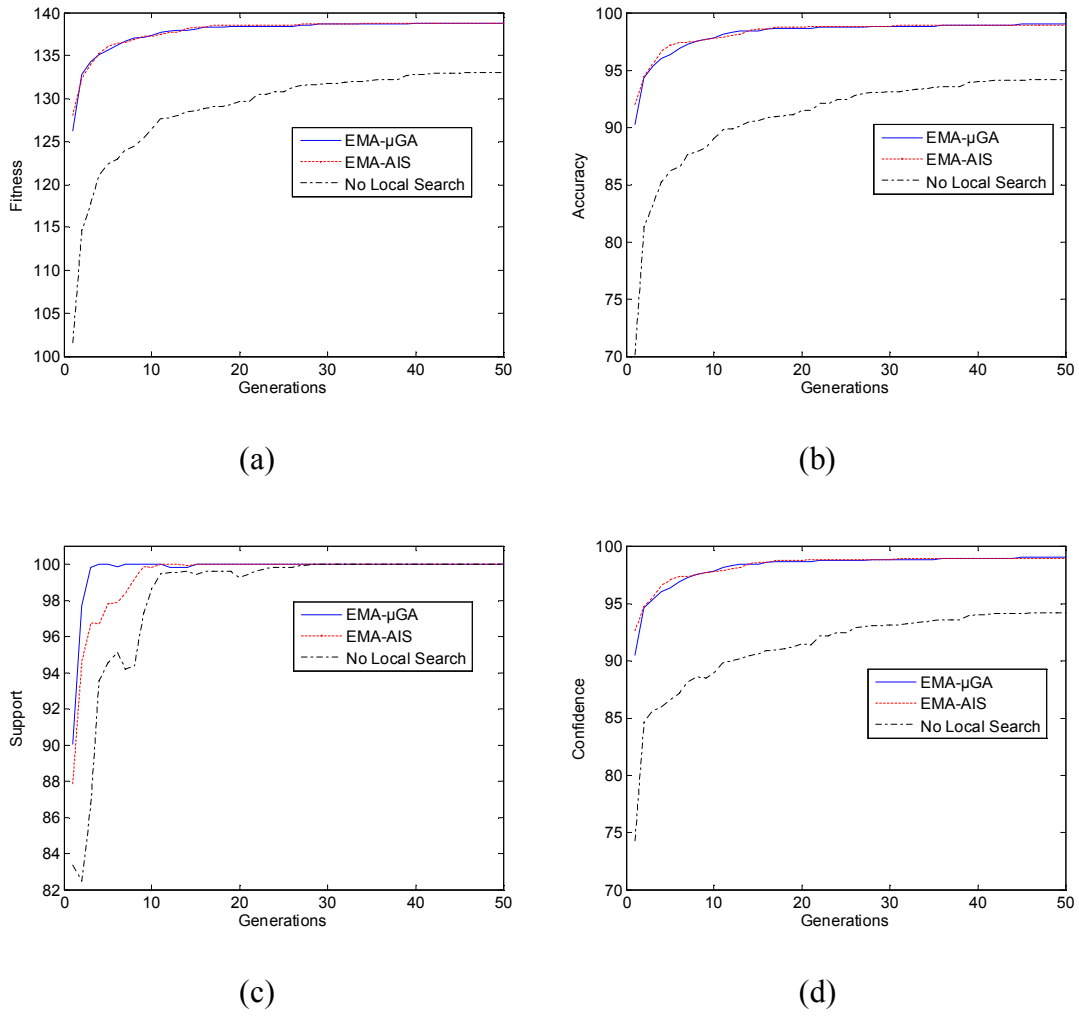


Figure 4.12. Training results for the iris data set

4.6.2 Rule Set Generated

The parent population rule sets are of different lengths and use different number of features within each rule due to the masking operator. Rule sets with larger number of rules do not guarantee better results than shorter rule sets. An appropriate number of rules within a rule set is required for good classification of the data sets.

An example of the rule set generated for the iris data set is given in Figure 4.13. Each rule contains different number of features. e.g., the first rule uses only the petal length while the second rule contains the sepal length, petal length and width. Each of these individual rules predicts a class and they collectively make up the whole rule set

and all the three classes of the iris problem are predicted. Only rule sets that predict all the classes of the problem are useful. If a rule set does not cover all the classes of the problem, some classes will be left unidentified. Further observations on the rule set show that not all the input features are required for decision making. The sepal width did not appear in the antecedent of any rules. This leads to the field of feature selection [30] which states that not all the given features of a data set are required for good classification. In fact, very often only a subset is required. The inclusion of all features might even deteriorate the performance of the algorithm as some features are detrimental and might interfere with the decision of the other features. This might imply that the sepal width is not required for good classification for the iris problem.

if	petal length $\leq$ 2.9575	then class is Setosa
else if	(sepal length $\leq$ 5.7553 or sepal length $\geq$ 5.9491) and (petal length $\geq$ 4.1471) and (petal width $\leq$ 1.0625 or petal width $\geq$ 1.6954)	then class is Virginica
else if	petal length $\geq$ 4.9726	then class is Virginica
else if	(4.7692 $\leq$ sepal length $\leq$ 6.6939) and (petal width $\geq$ 1.9455)	then class is Setosa
else if	2.8685$\leq$ petal length $\leq$ 5.2774	then class is Versicolour
else	class is Versicolour	

Figure 4.13: A rule set example for the iris data set

The number of rules generated by the algorithms over the runs are shown as box plots in Figure 4.14. Box plots [116][177] are used to present the median (line within the box), the lower quartile and upper quartile (given by the lower and upper

boundaries of the box) of groups of given data. The smallest and largest observations are also given in a box plot, with possible outliers marked by a cross.

EMA-μGA and EMA-AIS have fewer rules in a rule set for all problems compared to No LS. An appropriate number of rules is required rather than having a large rule set that does not guarantee better classification abilities. Without using LS, outliers are evident for all the three data sets. The differences between the median number of rules for EMA-μGA and EMA-AIS are not very great; 13.25 vs 12, 21.8 vs 21 and 7.15 vs 7.8 for the cancer, diabetes and iris data sets, respectively. Both EMA-μGA and EMA-AIS evolve rule sets with approximately similar complexity for all the data sets. The maximum and minimum number of rules within a rule set is smaller for EMA-μGA and EMA-AIS as compared to No LS. The incorporation of LS within the algorithm resulted in lower complexity rule sets yet achieving better performance.

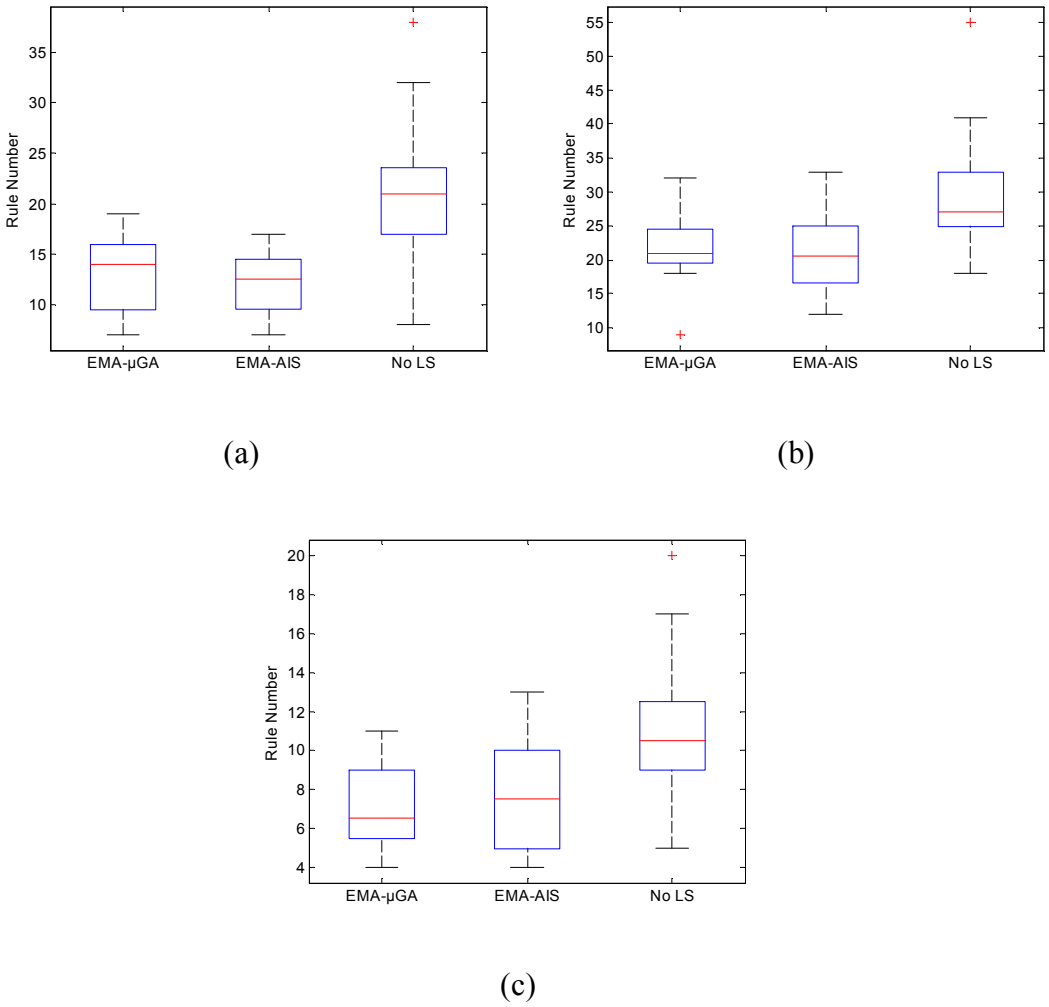


Fig 4.14: Average number of rules in a rule set (a) cancer (b) diabetes (c) iris

4.6.3 Results on Testing Data Sets

In this section, the support and generalization performance on the testing data sets are given. In addition, comparison of generalization accuracy is made with PART a rule-based algorithm in the literature [52].

4.6.3.1 Support on Testing Data Sets

The best training rule set is applied on the testing data set and Table 4.3 shows that the support on the testing data set is close to 100% for both the cancer and diabetes problems, and it is able to cover 100% of the instances for the iris problem.

The deviation on the number of instances that are supported is very small or zero. For every run, there is high confidence that the rule set evolved during the training phase is able to cover most of the testing data set instances and probably only one or two instances of the testing data set do not fit the rule set antecedents. When one rule set is able to support a high percentage of the testing data set, this requires only one rule set to be selected from the training phase. If each rule set has low support, several rule sets are needed to cover the entire testing data set.

Table 4.3: Support for testing data sets

		Mean support (%)	Standard deviation
Cancer	EMA – μGA	99.71	0.659
	EMA – AIS	99.86	0.255
	No LS	99.60	0.497
Diabetes	EMA – μGA	99.87	0.373
	EMA – AIS	99.95	0.160
	No LS	99.84	0.382
Iris	EMA - μGA	100	0.00
	EMA - AIS	100	0.00
	No LS	100	0.00

4.6.3.2 Generalization Accuracy

Since the support on the testing data set is almost 100%, in Table 4.4 which shows the generalization accuracy, it is certain that the generalization accuracy figures truly reflect the performance of the rule sets and the accuracy is also reflective of the confidence level.

Cancer Data Set: EMA-μGA and EMA-AIS obtained 97.4% and 97.6% mean generalization accuracy, respectively, and are the highest among the algorithms. All the algorithms proposed in this chapter are able to outperform PART in terms of mean

accuracy. EMA-μGA, EMA-AIS and No LS have the same maximum accuracy while No LS has lower minimum accuracy. No LS has higher standard deviation as compared to using LS. Though PART has the lowest standard deviation among all presented algorithms, it should be noted that the minimum accuracies for EMA-μGA and EMA-AIS are still higher than the mean accuracy of PART. Hence, for EMA-μGA and EMA-AIS, the deviation from the mean accuracy will not give results that are lower than PART.

Diabetes Data Set: PART obtained higher mean accuracy than No LS and has the lowest standard deviation. The algorithms proposed in this chapter have rather large standard deviations. The best algorithm among those proposed in this chapter is EMA-AIS, which has the highest mean accuracy yet lowest standard deviation. EMA-μGA is able to achieve the highest maximum accuracy at 80.2% which is a large difference from the mean accuracy of the rest of the algorithms.

Iris Data Set: The best algorithm on the iris data set is EMA-μGA where it obtained the highest mean, maximum and minimum accuracies. It also has lower standard deviation compared to No LS and PART. Both EMA-μGA and EMA-AIS have outperformed PART since they have higher mean accuracies with lower standard deviations.

Generally, across all data sets, the following observations are made:

- No LS has the lowest minimum accuracies and large standard deviations, hence LS is important for a more robust algorithm.
- EMA-μGA and EMA-AIS have the best mean accuracies.

Table 4.4: Generalization accuracy

		Mean accuracy (%)	Maximum accuracy (%)	Minimum accuracy (%)	Standard deviation
Cancer	EMA – μGA	97.414	99.425	95.402	0.942
	EMA – AIS	97.557	99.425	95.402	1.002
	No LS	96.236	99.425	93.678	1.532
	PART	94.9	-	-	0.4
Diabetes	EMA – μGA	75.052	80.208	71.875	2.292
	EMA – AIS	75.235	77.604	72.396	1.439
	No LS	73.229	77.083	68.75	2.133
	PART	74.0	-	-	0.5
Iris	EMA - μGA	97.297	100	94.595	0.877
	EMA - AIS	97.027	97.297	94.595	0.832
	No LS	94.595	100	83.784	3.921
	PART	93.7	-	-	1.6

4.7 Conclusions

A novel way of incorporating LS into EA for rule extraction is proposed in this chapter. Two LS algorithms, namely μGA LS and AIS inspired LS, are investigated and the results are analyzed and discussed in detail. The memetic algorithm is complimented by the use of variable length chromosome which naturally represents the rules for the problems to be solved. Several advanced variation operators are used to improve the algorithms. The algorithms are applied on real-world benchmarking problems and results show that both EMA-μGA and EMA-AIS have comparable performance. The results also show that LS is generally important for better efficiency as No LS is not able to attain the same level of performance as EMA-μGA and EMA-AIS. All the algorithms proposed in this chapter managed to get near 100% support level on the data sets.

Chapter 5

A Multi-Objective Rule-Based Technique for Time Series Forecasting

Classical methods like the exponential smoothing have been used as a simple method for Time Series Forecasting (TSF) [20][68][80][107][129]. Numerous statistical methods including the Box-Jenkins models, i.e., Autoregressive Moving Average (ARMA) model [28][75] have also been widely used. Though statistical methods provide satisfactory results, they are usually mathematically complex and cannot be easily applied by non-technical experts.

With the emergence of intelligent systems, computational intelligence techniques, like NN and EA, have gained much popularity in TSF. These techniques have been used extensively and also hybridized with classical and statistical methods to achieve improved results [8][82]. [171] proposed two enhancement factors, namely, density factor and distortion factor to the evolution process for financial time series segmentation. [35] uses a meta-level genetic algorithm for selecting the best ARMA model and a low-level GA for parameter optimization. The advantages of using NN for TSF lie firstly in its inherent ability to predict and model without the need of knowing the explicit details or underlying mechanism of the problem. Secondly, it is able to learn through repeated presentation of examples and handle non-linearity. However, the limitations of classical NN training lie in the use of gradient-based techniques which are more likely to get trapped in local optima [7].

In early Evolutionary Neural Networks (ENN) research, EA have been used to

evolve the weights and architectures of NN in TSF. More recently, there are other interesting works on ENN for TSF that consider other aspects [31][36][110]. [47] hybridized a NN and a modified GA, where the evolutionary process searches for the required minimum time lags in representing the series. Other interesting aspects of EA and its hybrids for TSF also surface in view of the great potential. These works are orientated towards using rules within an evolutionary framework for TSF [42][103][112]. Apart from these works on extracting knowledge from TSF, there are rarely any prominent works.

This chapter proposes a Multi-Objective Rule-Based Technique (MORBT) which intends to add on to the limited number of works in the literature on evolving rules for TSF, opening up new avenues and interesting possibilities on applications of EA. A dual phase approach is adopted, with phase I concentrated on evolving a population of general regression coefficients for the data set. The coefficients are optimized by the use of EA instead of being calculated, e.g., by statistical methods like Least-Squares Analysis (LSA). LSA requires several statistical assumptions, involves highly complex mathematics and is not suitable in all cases, e.g., in the presence of singular matrices whose inverse cannot be calculated accurately. Therefore, the use of EA is less restrictive in these cases. In phase II, a rule-based approach is used to locate the optimal regions of the series, for which each regression coefficient string is suited. Instead of using a set of general regression coefficients throughout, several sets of regression coefficients are identified. In phase II, each chromosome is coupled with a boundary, operator and regression coefficient string. The motivation behind the dual phase approach is to break down the otherwise large search space of the difficult combinatorial optimization problem of searching for both the optimal set of regression coefficients and rule parameters.

The two important aspects in evolving a rule are to obtain a low predictive error and a high coverage in the objective space. However, these two conflicting objectives are rarely handled well. Multi-Objective Evolutionary Algorithms (MOEA) are able to present solutions to a problem with several objectives in the form of alternate trade-offs and are able to evolve multiple Pareto solutions concurrently and obtain a Pareto optimal set in a single run [58][73][91][155][190]. MOEA would act as a suitable method to evolve solutions with low error and high rule coverage. With these in mind, phase II of the proposed algorithm uses MOEA and the concept of Pareto dominance to optimize the two objectives of low error and high coverage simultaneously.

Section 5.1 presents the basics of multi-objective optimization. The details of the proposed features and operators of the MORBT are described in Section 5.2 while the MORBT overview is given in Section 5.3. Experimental setup and data sets used are described in Section 5.4. Studies upon financial time series data sets are conducted in Section 5.5. Conclusions are subsequently drawn in Section 5.6.

5.1 Multi-Objective Optimization

Multi-Objective (MO) problems refer to problems with more than one objective to be optimized. Taking a multi-objective minimization problem as an example, the aim is to minimize the objective set, $O(X)$, where $O(.) = \{o_1, o_{2,}, ..., o_n\}$ are the n objectives and $X = \{x_1, x_2, ..., x_m\}$ is the m dimensional parameter vector [157][158].

Real-world optimization problems often entail optimizing simultaneously various objectives which are usually conflicting and non-commensurable. Most existing literature concentrates on one objective and ignores the other, or considers them concurrently using a weighted sum approach. However, the objectives to be considered are very often non-commensurable to be placed on the same platform.

Moreover, if the objectives are conflicting, it is not acceptable to present solutions in a single-objective manner, but should be in a form of trade-offs [51][81][155].

Solutions to multi-objective optimization can be casted in a Pareto optimal set. Figure 5.1 illustrates an example of the solutions obtained for a two-objective minimization problem. The Pareto optimal set (marked by cross) is the set of non-dominated solutions representing the tradeoffs between the objectives. When a solution is not dominated it means there is no other solution that has better performance in both the objectives or there is no other solution that is better in one objective and equal performance for the other objective. Within the Pareto optimal set, when a solution point is identified, in order to pick another solution that can improve in one of the objectives, this has to be done at the expense of the other objective [41][51][145][155][157]. Let $A=[a_1,a_2]$ and $B=[b_1,b_2]$ be two solutions of a two-objective minimization problem. Table 5.1 shows the dominance relationships between them.

Table 5.1: Dominance relationships

Objective 1	Objective 2	Outcome
$a_1 < b_1$	$a_2 < b_2$	A strongly dominates B
$a_1 < b_1$	$a_2 \leq b_2$	A weakly dominates B
$a_1 \leq b_1$	$a_2 < b_2$	A weakly dominates B
$a_1 > b_1$	$a_2 < b_2$	Equal dominance
$a_1 < b_1$	$a_2 > b_2$	Equal dominance
$a_1 = b_1$	$a_2 = b_2$	Equal dominance
$a_1 > b_1$	$a_2 > b_2$	B strongly dominates A
$a_1 \geq b_1$	$a_2 > b_2$	B weakly dominates A
$a_1 > b_1$	$a_2 \geq b_2$	B weakly dominates A

To find the Pareto optimal set, MOEA are often deemed as a promising methodology due to its global search capability. It evaluates several candidate

solutions simultaneously within a generation and through the use of a combination of guided (fitness selection) and stochastic search (variation operator) processes, it is able to locate the global optimum. This intrinsic property enables a macro-view of the problem, allowing the algorithm to find a diverse set of solutions that is close to the optimal Pareto front [59].

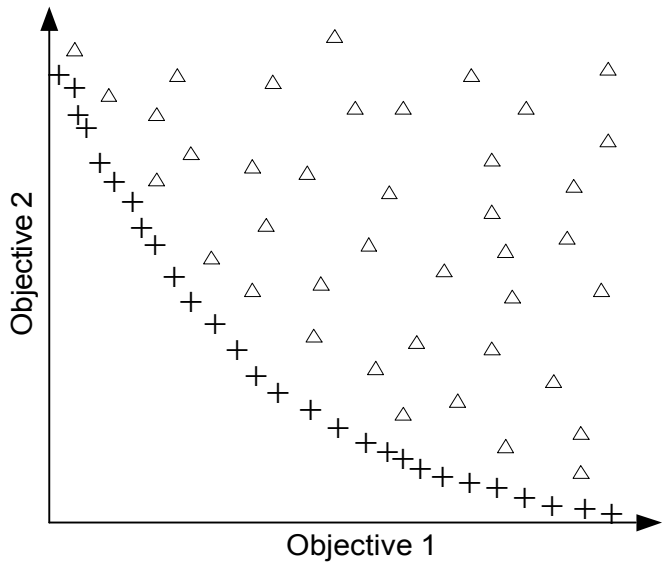


Figure 5.1: Pareto front

5.2 Details of the Multi-Objective Rule-Based Technique

The following sections will present the features and operators for MORBT.

5.2.1 Initialization and Chromosome Representation

A population of real-value regression coefficients is being initialized in phase I and the values of the coefficients are randomly generated in the range of [-0.5…0.5]. The length of the coefficient chromosomes is fixed and reflects the sliding window length.

In phase II, the regression coefficient strings are integrated with the boundary and operator strings. Each chromosome consists of three strings of genes (Figure 5.2)

which uses the Michigan encoding scheme. The first string encodes the boundary conditions for each independent input observation within the sliding window and is represented by two real-value coded genes. One encodes the lower boundary while the other encodes the upper boundary. The lower boundary for the *i*th observation is denoted by L_i while the upper boundary is denoted by U_i. This boundary value gives the numeric threshold for the observation and is to be used concurrently with the operator string. The second string encodes the operators which indicate the equality and inequality operators to be applied on the boundary string. The four operators encoded are namely, greater than or equal "≥", less than or equal "≤", within a range of "≤ ≤", and less than or greater than"≤ ≥", given by $x \geq L$, $x \leq U$, $L \leq x \leq U$ and ($x \leq L$ or $x \geq U$), respectively, where x is the data set numeric value. The third string encodes the regression coefficients.

The length of a chromosome in phase II is also determined by the sliding window length, *w*. The whole chromosome would be interpreted as samples that fit the antecedents of the rule would have its output determined by the associated regression coefficients.

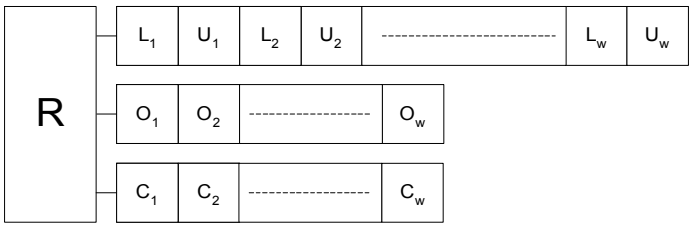


Figure 5.2: Genotype representation of a chromosome

5.2.2 Error Function

The common evaluation metric for TSF algorithms would be the difference between the actual and forecasted observations, i.e., the error term *E*, and the

objective would be to minimize *E*. Several variants of the error term are given and studied in the literature [9]. The more common ones are stated as:

$$\text{SSE} = \sum_{t=1}^{N} (d_t - \hat{x}_t)^2 \qquad (5.1)$$

$$\text{MSSE} = \frac{1}{N} \sum_{t=1}^{N} (d_t - \hat{x}_t)^2 \qquad (5.2)$$

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{t=1}^{N} (d_t - \hat{x}_t)^2} \qquad (5.3)$$

where SSE = Sum of Squared Error, MSSE = Mean Sum of Squared Error and RMSE = Root Mean Squared Error, d_t is the actual observation, $\hat{x}_t$ is the predicted value and is calculated as $\hat{x}_t = X \bullet C^T$, $X = [x_{t-w} \ldots . x_{t-1}]$ are the input values and $C = [c_1 \; c_2 \ldots . . c_w]$ are the regression coefficients. *N* is the total number of values to be forecast.

5.2.3 Tournament Selection

Binary tournament selection with replacement is used for selection of parents for offspring creation in phase I. Two parents are randomly selected and the parent with the higher fitness level would be taken as a snapshot for offspring creation. The two parents are then replaced back into the population. The selection of the second parent would then follow the same procedure. After the two parents are selected, crossover is carried out.

5.2.4 Crossover

A single random point crossover operation is applied in phase I on the coefficient strings. The crossover operator allows parent chromosomes to exchange useful information between them in order to create new chromosomes. The crossover operation also allows new solution space to be explored with the creation of new chromosomes. An example of the crossover operation is shown in Figure 5.3.

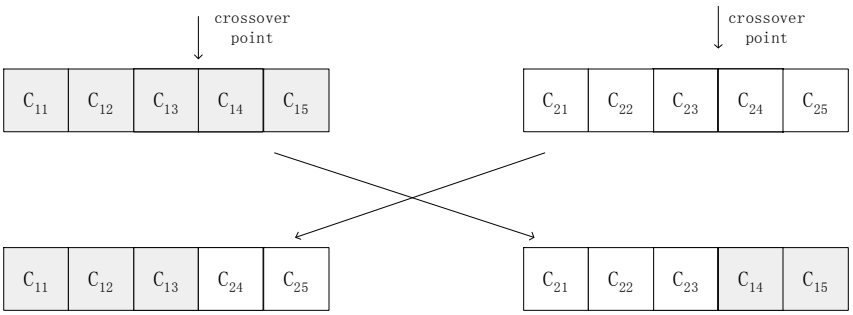


Figure 5.3: Single random point crossover

5.2.5 Mutation

The mutation operator ensures that new areas in the search space are being explored. The probability of mutation is important in the evolution process. If the mutation probability is small, the mutated chromosome would not be very different from its original structure, and is less likely to escape from local optima. On the other hand, having large mutation probability directs the search towards random search and increases the possibility of disrupting the structure of chromosomes that may carry good solutions.

Phase I mutation takes place on the coefficient string while in phase II, mutation is applied to the boundary and operator strings as well. Real-value mutation is used and a random value between $\pm 10\%$ of the original value is added to the alleles of the boundary strings. Constraints are set on the boundary strings and at any time if the

value of the lower boundary is higher than the value of the upper boundary after mutation, these two values are swapped.

5.2.6 Multi-Objective Pareto Ranking

Before the use of multi-objective optimization algorithms, multi-objective problems are usually translated into single-objective optimization problems [145], which makes the problem simple and easy to handle. However, such techniques like weighted average of the different objectives are not natural and often compromise certain properties of the various objectives. In addition, the use of the weighted averaging introduces other issues like the assignment of the weights which is related to the scaling of the different objectives.

The two objectives to be optimized in this chapter are conflicting. Rules with higher coverage are able to capture more time series sequences. Higher coverage rules are more general rules compared to lower coverage rules. Lower coverage rules are specific rules that cater to a limited number of time series sequences, therefore they are expected to have lower mean error than the more general rules. The formalization of these two objectives is given as:

$$f_1 = \min\left\{\sqrt{\frac{1}{n}\sum_{j=1}^{n}(d_j - \hat{x}_j)^2}\right\}, \quad n = \sum_{i=1}^{N} Z_i \tag{5.4}$$

$$f_2 = \max\left\{\frac{n}{N}\right\} \tag{5.5}$$

The first objective function f_1 is to minimize the RMSE, where n is the total number of samples that fits the rule and N is the total number of samples in the data set. $Z_i = 1$ if the rule covers the sample at time t_i and $Z_i = 0$ if it does not. The second objective f_2 is to maximize the coverage (number of samples that fit the antecedents of

the rule).

The multi-objective rule optimization problem lies in finding solutions that fit the conflicting objectives of maximizing the rule coverage and minimizing the error. Figure 5.4 shows an example solution for optimizing these two conflicting objectives. When the rule coverage is increased, the error is increased. The solutions indicated by the crosses form the Pareto optimal set while those indicated by triangles are the dominated solutions.

There are several types of Pareto ranking schemes available in the literature and the one used in this chapter is given as:

$$i_{rank} = 1 + d_i \tag{5.6}$$

where i_{rank} is the rank of the ith solution and d_i is the number of solutions that dominates the ith solution. This equation results in all non-dominated solutions having the rank 1 and there might be some ranks not being occupied [51][59][156][157].

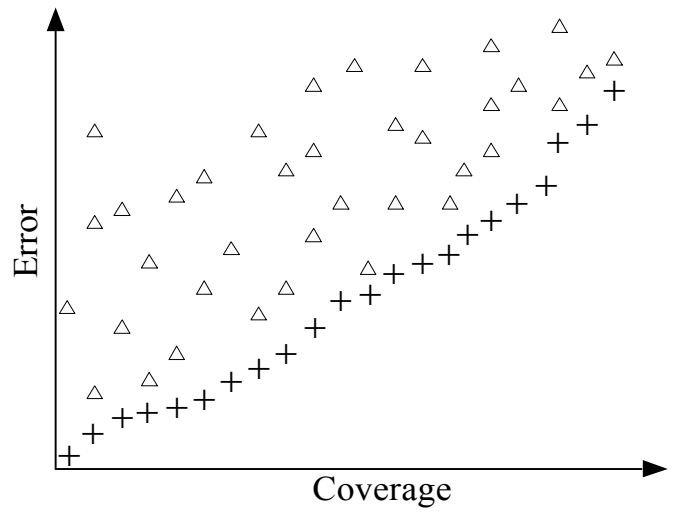


Figure 5.4: Tradeoffs between training error and coverage

Apart from optimizing the various objectives of the problems, MOEA would also want the Pareto optimal set to be as wide-spread and well-distributed as possible. This ensures that crowding would not occur and a diverse set of solutions can be obtained to provide a wider selection choice for the users [40].

In the context of this chapter, the solutions can only occupy certain positions on the Pareto front. This is because the coverage function is based on the number of samples covered by the rule which would give a limited number of values. Crowding of the values has much lower possibility unlike in the case of real-value solutions, where solutions can occupy infinite positions. Hence, the diversity issue is not a problem here (the experimental results in later section will illustrate this point clearer).

5.2.7 Fine-Tuning

The coefficients of the rules are fine-tuned at every T generations. Since the coverage is not affected by this process, the effect of fine-tuning is to lower the prediction error and push the Pareto front downwards (Figure 5.5). This fine-tuning process is only applied to the rank 1 solutions. Each rank 1 chromosome is first replicated to create a population of offspring which would undergo real-value mutation. The chromosome with the lowest error would emerge as the parent. This process repeats for a predefined number of iterations.

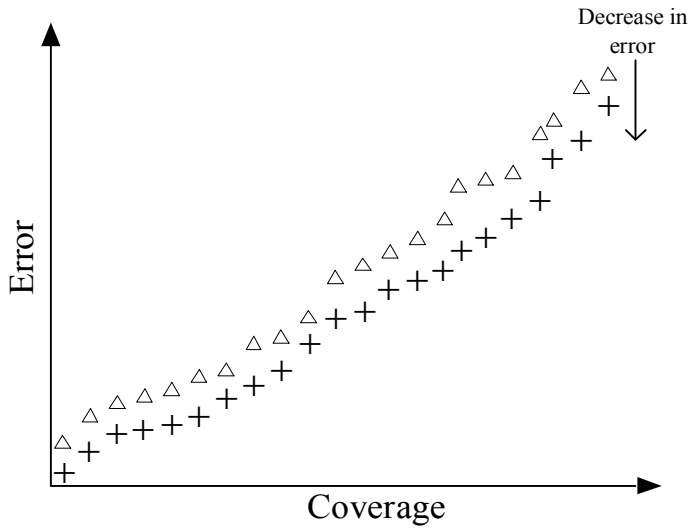


Figure 5.5: Resulting Pareto front from fine-tuning

5.2.8 Elitism

In phase I, elitism is implemented by combining the parent and offspring population and choosing the better chromosomes to be the parents for the next generation. In phase II, the process is implemented by combining the parents and offspring together and selecting the non-dominated solutions. The size of elitism is dependent on the number of non-dominated solutions distributed along the front. Elitism has the effect of improving convergence speed.

5.3 Multi-Objective Rule-Based Technique Overview

The overall training process is separated into two stages to breakdown the task of optimizing both the coefficients and rule parameters together. The pool of optimized regression coefficients from phase I is smoothly transited to phase II to combine with the rule parameters. The overviews of phase I and II are given in Sections 5.3.1 and 5.3.2.

5.3.1 Phase I: Algorithm Overview

The design process begins with random initialization of a population of chromosomes representing the forecasting regression coefficients. Binary tournament selection is used to select parents for crossover. Offspring creation using crossover based on P_c (crossover probability) is carried out on the parents. The newly created children undergo real-value mutation based on P_m (mutation probability) in order to introduce diversity within the population. The fitter chromosomes among the children and parent populations are brought over to the next generation as the new parents. The training continues until the maximum generation is reached. At the end of phase I, a set of general regression coefficients for the training data set is obtained. The coefficient string is now near optimal for general prediction. The flow chart of phase I is given in Figure 5.6.

The aim of phase I is to evolve the regression coefficients instead of calculating it using statistical techniques. Using EA to evolve the coefficients has the advantage of being more flexible and not requiring complex mathematical technicalities. There are not many constraints and assumptions that EA needs to adhere. For example, when using least-squares analysis, a linear regression problem can be represented in the form of $Y = \beta X + \varepsilon$, where Y is the dependent variable vector, X is the independent variable matrix, β is the coefficient vector and ε is the error vector. If it satisfies the Gauss-Markov assumptions, then the parameter β is calculated as $\beta = (X^T X)^{-1} X^T Y$, however this requires matrix $(X^T X)$ to be non-singular.

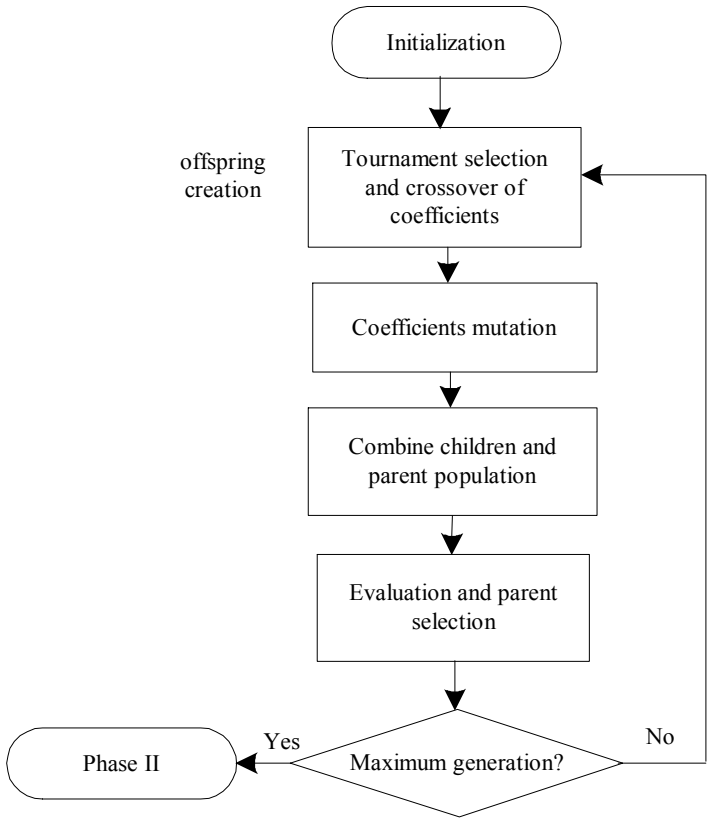


Figure 5.6: Flow chart of phase I

5.3.2 Phase II: Algorithm Overview

Based on the pool of chromosomes with coefficients optimized, the boundary conditions and operators are evolved in phase II together with the incorporated coefficients. Each chromosome within the population represents a rule which covers a predefined interval of the time series. Benefiting from the already evolved general coefficients from phase I, the initial large search space of phase II is narrowed. If the boundary conditions, operators and coefficients are evolved simultaneously from initialization, the search space would be extremely large and multi-modal.

The first step in phase II is to initialize a population of boundary conditions and operators. These chromosomes are evaluated and Pareto ranked. Since the rank 1 chromosomes are the best solutions obtained, these are taken as the parent population.

These rank 1 chromosomes are replicated to create a pool of offspring. Though the replication of the rank 1 chromosomes is similar to the process of Multi-Objective Artificial Immune Systems (MOAIS) [34][101][102] method of proliferation, it is viewed as a simple and efficient method to create offspring rather than to emulate MOAIS. The associated rule boundary, operator and coefficient strings of the offspring are then mutated. The mutated offspring are evaluated based on the two objective functions, i.e., coverage and normalized RMSE. The parent population is combined with the offspring population and the solutions in the combined population are re-ranked based on their dominance relationship. Ranked 1 chromosomes would be used as parent chromosomes for the next generation. If there are more parents than the predefined front limit, truncation process would be carried out. Chromosomes with the lowest error are selected and the rest are discarded. Therefore, the size of the parent population is dependent on the number of individuals distributed along the Pareto front as well as the front limit. The coefficients of the parents go through fine-tuning at every T generations ($T = 5$ is used). The overall process continues until the stopping criterion, which is the maximum number of generations, is reached.

Evolution starts from limited points in the search space and expands to cover an increasingly wider area. Specific rules cover fewer samples and are more accurate; on the other hand, general rules cover larger number of samples at the expense of lower training accuracy but are important as they increase the possibility of covering a larger solution space of the testing data set. If all the rules are specific, there are high chances that they would only fit the observations of the training data set and most of the test data would not be captured by the rule. The flowchart of phase II is shown in Figure 5.7. At the end of the evolution process, there might be some samples of the training data set which are not covered by the training rules and general coefficient

string would be used on these samples.

5.3.3 Testing Phase

At the end of the training process, all parent chromosomes are arranged in ascending order according to error, to be applied on the testing data set. The first rule that is able to cover the presented sample will be used for prediction. If there are no rules that are able to cover this sample, the general coefficient string from phase I will be used.

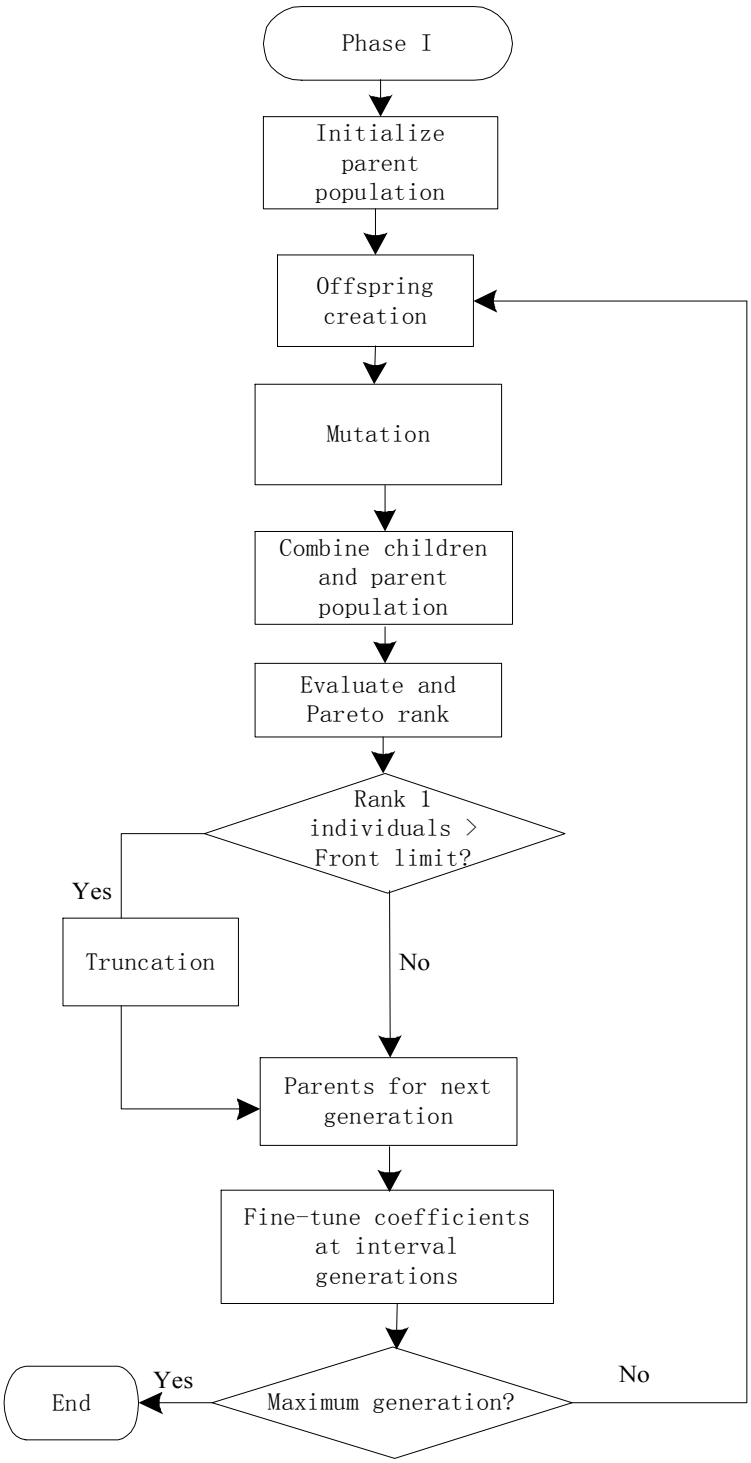


Figure 5.7: Flow chart of phase II

5.4 Experimental Setup and Data Sets

MORBT is implemented using the MATLAB technical computing platform and the corresponding experiments are performed on Intel Pentium 4 2.8 GHz computers. The parameter settings for MORBT and the details of the data sets used are given in the following sections.

5.4.1 Experimental Setup

The number of generations in phase I is only half of phase II and the parent to offspring ratio is also smaller. In phase I, for every 1 parent there are 6 offspring but in phase II, for every 1 parent there are 10 offspring. This is because the motivation of phase I is to find an initial pool of good regression coefficients without requiring them to be fully optimized. In phase II, these coefficients would be further evolved concurrently with the rule parameters. In addition, there is only one parameter in phase I, which is the regression coefficient string, to be optimized, while in phase II there are three parameters - the regression coefficient, the boundary and the operator strings. Apart from having more parameters to be optimized in phase II, there is an additional objective, which is the coverage, to be considered.

As the Front Limit (FL) is a crucial parameter for the coverage on the training data sets, two front limit sizes are given in phase II as different experimental setups to investigate the effects of the FL sizes on the algorithms. One setting limits the number of individuals allowed on the Pareto front to be 50 while the other doubles the value to 100 individuals. Both phases use the same mutation probability and mutation size for the regression coefficients. These parameter settings are used for all the data sets in this experiment.

Table 5.2: Parameter settings for MORBT

	Phase I	Phase II
Populations	Parent: 10 Offspring: 60	Parent: Rank 1 Offspring: x10
Chromosome type	Real-number representation	Rule-based
Selection for parent	Fitter chromosomes	Rank 1 individuals
Generations	30	60
Mutation probability	0.6	0.6
Mutation	Real-value	Real-value and randomly generated operators
Front limit	N.A.	50, 100

The MORBT is compared to algorithms that use only a single phase.

GA: Comparisons are made with genetic algorithms that involves phase I of the MORBT only, i.e., the algorithm uses the coefficients only and not the rules.

SP-MORBT: Comparisons are made with an algorithm that uses only phase II of the MORBT. This algorithm evolves all the parameters together without using an initial pool of good coefficients. It is termed Single Phase MORBT (SP-MORBT).

These comparisons are necessary to show the importance of having a rule-based and dual phase approach. The number of generations for the single phase algorithms are more than the MORBT to compensate for the lack of dual phase. The parent and offspring populations are also larger for GA as compared to Phase I of the MORBT. For the comparisons to be consistent, other parameter settings of these two algorithms are the same as MORBT.

Table 5.3: Parameter settings for GA

	GA
Populations	Parent: 30 Offspring: 150
Chromosome type	Real-number representation
Selection for parent	Fitter chromosomes
Generations	150
Mutation probability	0.6
Mutation	Real-value

Table 5.4: Parameter settings for SP-MORBT

	SP-MORBT
Populations	Parent: Rank 1 Offspring: x10
Chromosome type	Rule-based
Selection for parent	Rank 1 individuals
Generations	100
Mutation probability	0.6
Mutation	Real-value and randomly generated operators
Front limit	100

5.4.2 Data Sets

The Financial Times Stock Exchange (FTSE) of the London stock exchange, the Standard and Poor's 500 (S&P 500) and the National Association of Securities Dealers Automated Quotations (NASDAQ) of the United States market are used. For each index, weekly data for a ten-year period is used [184]. These are actual real-world data collected from the last day closing index of the trading week. 70% of the data are used as training data set while the remaining 30% are used as the testing data set.

These data sets are normalized in the range of [0,1] for training purposes. The normalization method used is:

$$x_{ni} = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} \tag{5.7}$$

where x_{ni} is the normalized *i*th data, x_i is the *i*th data to be normalized, $x_{\max}$ and $x_{\min}$ are the maximum and minimum values, respectively, in the training data set.

5.5 Experimental Results and Analysis

The experimental results showcasing the different aspects of findings are studied in detail in the following sections.

5.5.1 A Rule Example

Figure 5.8 shows an example of the rule evolved for the FTSE. The values shown are the normalized values extracted from the training phase. Different operators are used for the input observations. As the values of the data sets are highly fluctuating, there are large differences among the input observation boundaries. The coefficients reveal the relative importance of each input observation to forecast the output value. Those with larger coefficients have higher importance while those with smaller coefficients are of lower importance. This rule shows that the later observations (observations closer to the value to be predicted) have larger coefficients. This observation is logically intuitive as one would expect the more recent values being more indicative of the future values.

if $0.096 \le x_{i1} \le 0.53375$ and $0.29236 \le x_{i2} \le 1$ and ($0.22255 \ge x_{i3}$ or $x_{i3} \ge 0.8699$) and ($0.3447 \ge x_{i4}$ or $x_{i4} \ge 0.5396$) and ($x_{i5} \ge 0.36038$)

then $y_i = -0.043895x_{i1} + 0.12684x_{i2} - 0.072595x_{i3} + 0.20471x_{i4} + 0.2120x_{i5}$

where x_{ij} is the *j*th input observation value for the *i*th sample and y_i is the predicted output value for the *i*th sample.

Figure 5.8: A rule example for the FTSE

5.5.2 Algorithm Coverage

The coverage of the rules evolved on the training and testing data sets are presented in this section. Table 5.5 and Table 5.6 present the results of MORBT using FL = 100 and FL = 50, respectively. When front limit of 100 is used, the mean coverage for all training data sets is over 99%. This shows that only a few samples are not covered by the evolved rules. From the maximum and minimum coverage, the deviation of samples being covered is low, with less than 5% deviation; this is also supported by standard deviation results. In addition, the mean, maximum and minimum coverage on all testing data sets for all runs are 100%, and hence the standard deviation is 0. Since the coverage of the samples is high, the error results presented in the following sections (using FL = 100) would be reflective of the rule performance, i.e., the samples use coefficients assigned by the rules rather than the general coefficient from phase I.

The mean coverage of MORBT on the training data set has dropped when the FL is reduced to 50. The drop in mean coverage is most significant for the NASDAQ problem. This indicates that a FL of 50 would not be adequate to cover all samples. Though the maximum coverage across the runs on the FTSE and S&P 500 managed to maintain at 100%, the minimum coverage has decreased. There are large fluctuations on the rule coverage causing large standard deviations. The same

observations are seen on the testing data set. MORBT, with FL = 50, on the NASDAQ testing data set has the lowest mean, maximum and minimum coverage, and the largest SD.

Table 5.5: Rule coverage - MORBT with FL = 100

	Coverage on training data				Coverage on testing data			
	Mean (%)	Max (%)	Min (%)	SD	Mean (%)	Max (%)	Min (%)	SD
FTSE	99.92	100	99.17	0.264	100	100	100	0
S&P 500	99.36	100	96.67	1.032	100	100	100	0
NASDAQ	99.28	100	96.67	1.008	100	100	100	0

Table 5.6: Rule coverage - MORBT with FL = 50

	Coverage on training data				Coverage on testing data			
	Mean (%)	Max (%)	Min (%)	SD	Mean (%)	Max (%)	Min (%)	SD
FTSE	87.25	100	53.89	19.78	90.45	100	63.69	15.50
S&P 500	90.72	100	77.22	7.20	98.21	100	93.59	2.43
NASDAQ	73.97	82.5	65.56	5.60	78.66	99.36	33.12	22.52

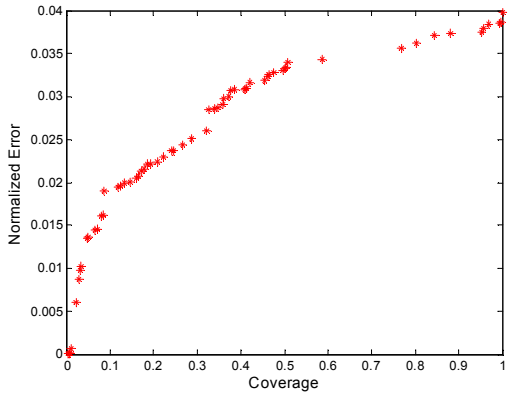
Table 5.7 presents the rule coverage results for SP-MORBT with FL = 100. With the use of single phase approach, the algorithm is still able to obtain more than 90% coverage on the training data set and a consistent 100% coverage on the testing data set. The results have very low standard deviation on the training data set and zero standard deviation on the testing data set. It is concluded that the FL has a large effect on the coverage of the training samples, which in turn affects the coverage on the testing samples.

Table 5.7: Rule coverage - SP-MORBT with FL = 100

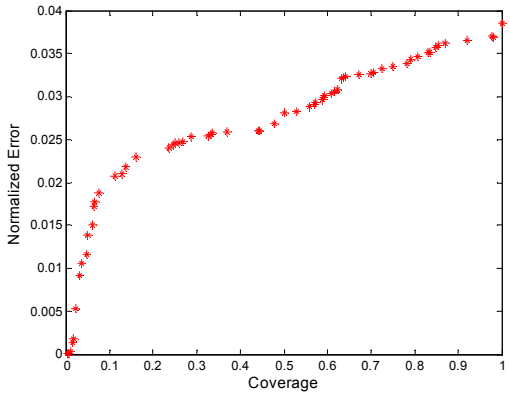
	Coverage on training data				Coverage on testing data			
	Mean (%)	Max (%)	Min (%)	SD	Mean (%)	Max (%)	Min (%)	SD
FTSE	100	100	100	0	100	100	100	0
S&P 500	99.89	100	98.89	0.3514	100	100	100	0
NASDAQ	98.61	100	94.72	1.5876	100	100	100	0

5.5.3 Pareto Front

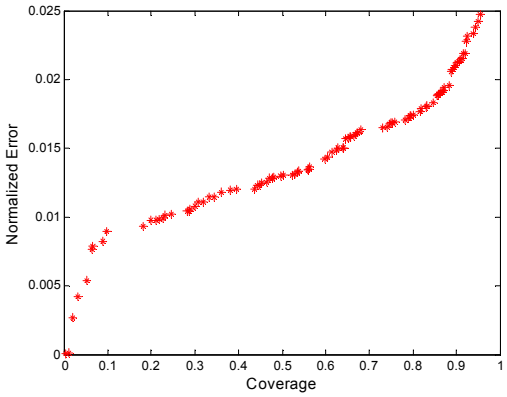
Figures 5.9 and 5.10 show the non-dominated Pareto front evolved by phase II of the MOBRT using FL = 100 and FL = 50, respectively. Both figures show relatively well-spread fronts rather than fronts that crowd at particular points of the solution space. The number of points show that there are many solutions being evolved for each problem and MORBT is also able to find many diverse rules. By decreasing the front limit, the number of solutions is sparser over the front. Figure 5.11 shows the front obtained using SP-MORBT. Results show well-spread fronts with many solutions.



(a) FTSE

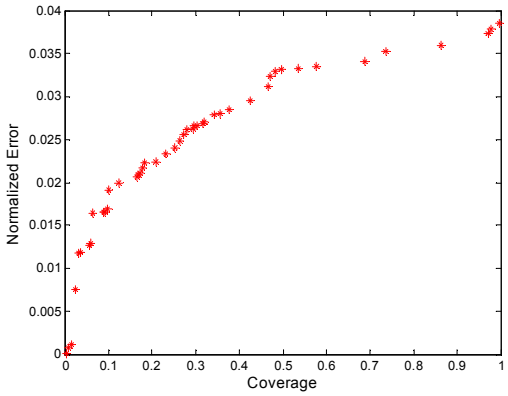


(b) S&P 500

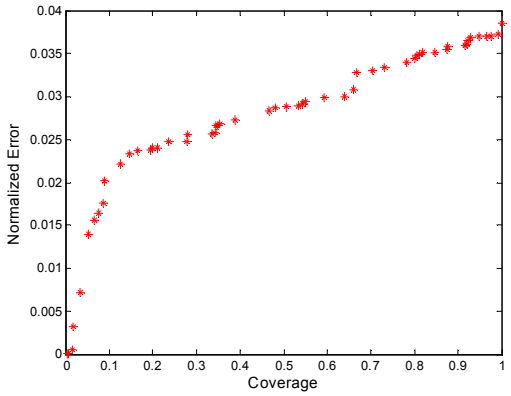


(c) NASDAQ

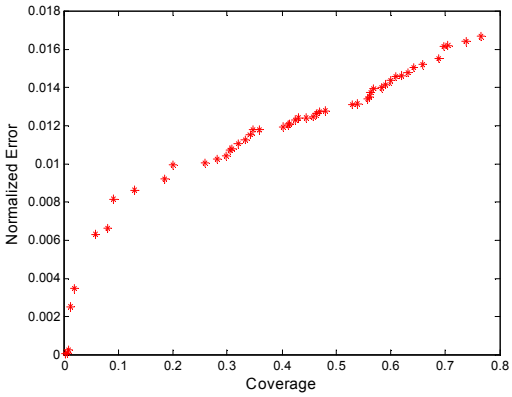
Figure 5.9: Rank 1 Pareto front - MORBT with FL = 100



(a) FTSE

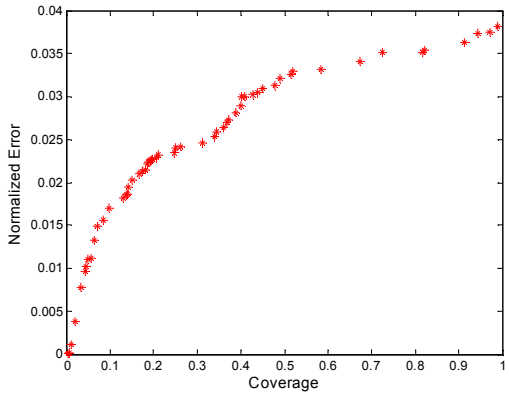


(b) S&P 500

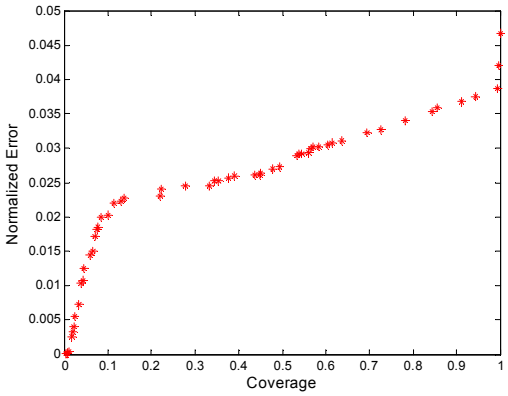


(c) NASDAQ

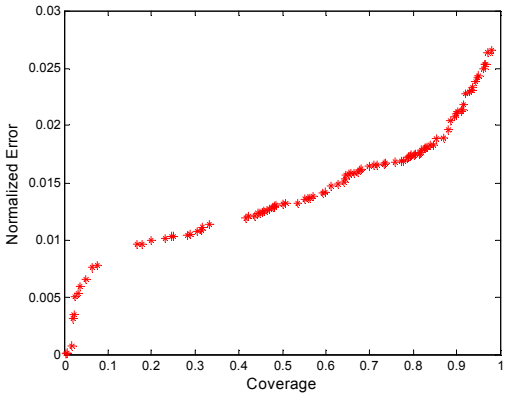
Figure 5.10: Rank 1 Pareto front - MORBT with FL = 50



(a) FTSE



(b) S&P 500



(c) NASDAQ

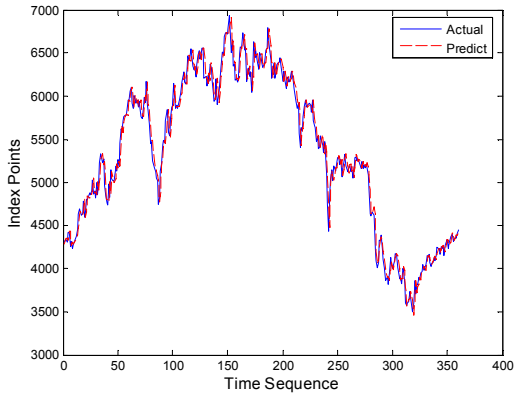
Figure 5.11: Rank 1 Pareto front - SP-MORBT with FL = 100

5.5.4 Actual and Predicted Values

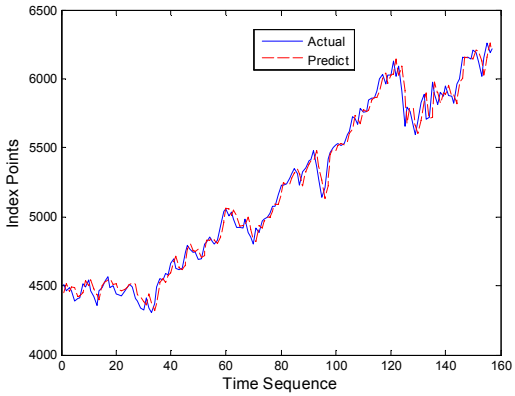
Even though the stock indexes manifest itself as highly fluctuating and unpredictable (Figures 5.12 – 5.14), the MORBT is still able to predict quite closely on the training and testing data sets. As shown in the figures, the training data set spreads over a larger range than the testing data set, meaning there is a possibility that only a portion of the rules of the training data set is being used on the testing data set. From the figures, it is hard to see which algorithm actually performs better. Closer examinations would be made in the next section.

5.5.5 Generalization Error

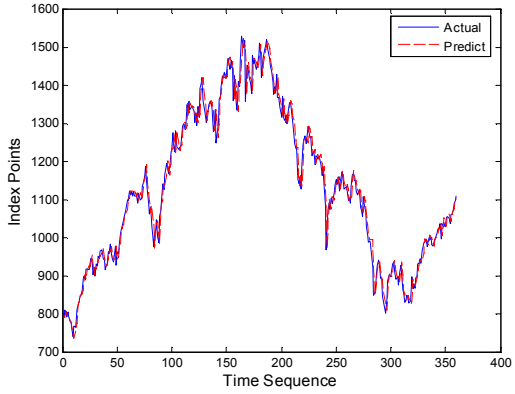
The results presented in Table 5.8 and Table 5.9 show the actual value (not normalized) of the generalization RMSE. The mean results obtained by averaging across all runs show that MORBT (FL=100) has lower RMSE than MORBT (FL=50) for all data sets. In addition, MORBT (FL=100) has lower maximum error and smaller standard deviation than MORBT (FL=50) for the FTSE and NASDAQ. The poorer results of MORBT (FL=50) could be due to the lack of coverage. Therefore, it is important for most of the samples to be covered by the rules.



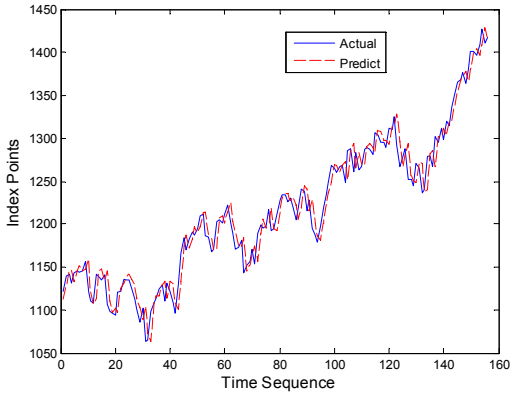
(a) FTSE training data



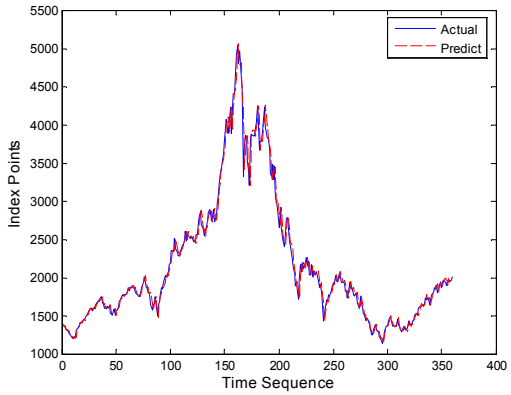
(b) FTSE testing data



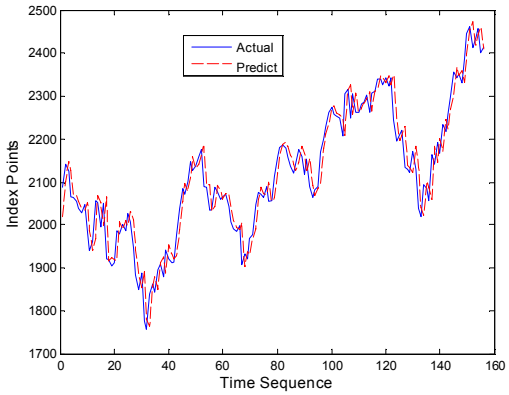
(c) S&P 500 training data



(d) S&P 500 testing data

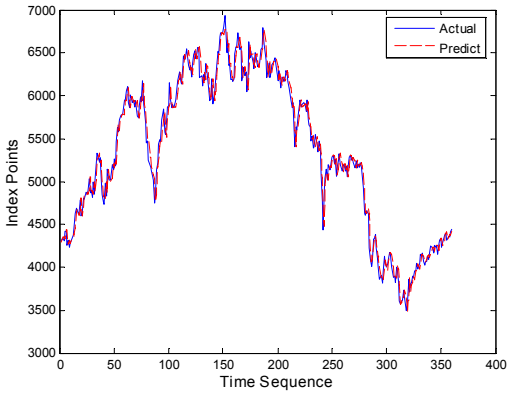


(e) NASDAQ training data

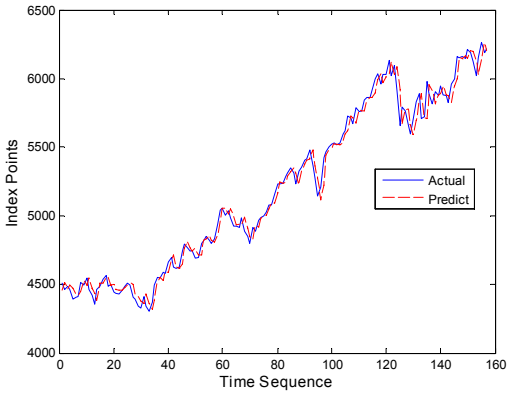


(f) NASDAQ testing data

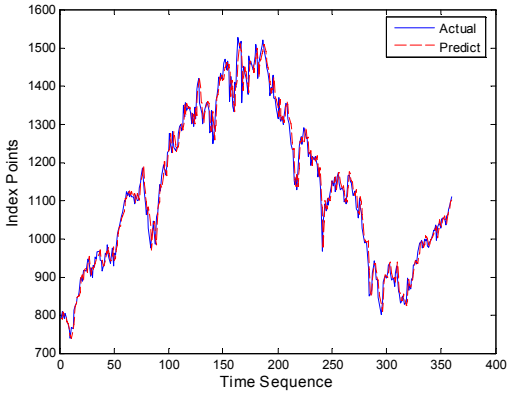
Figure 5.12: Training and testing data sets prediction - MORBT with FL= 100



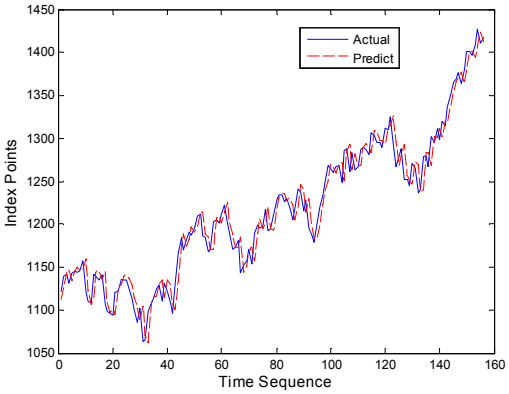
(a) FTSE training data



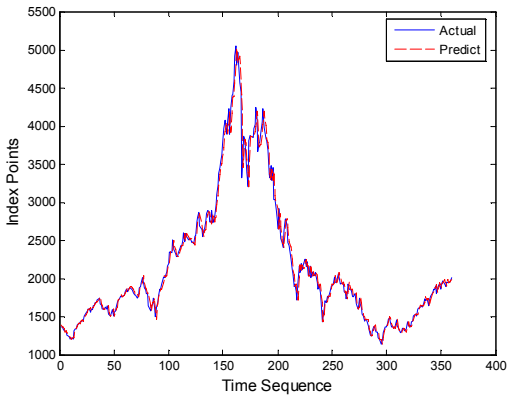
(b) FTSE testing data



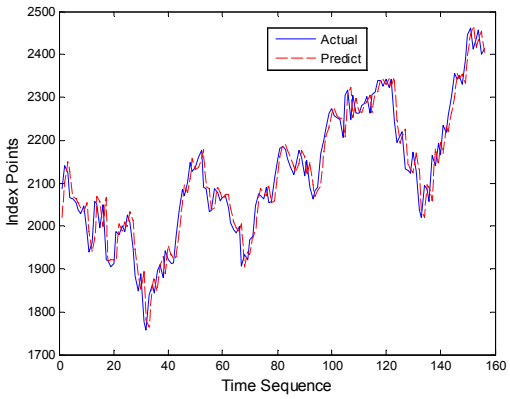
(c) S&P 500 training data



(d) S&P 500 testing data

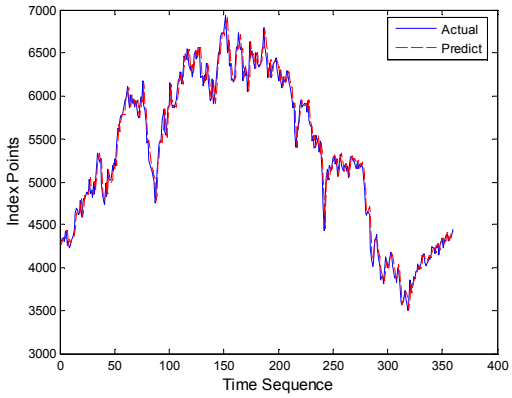


(e) NASDAQ training data

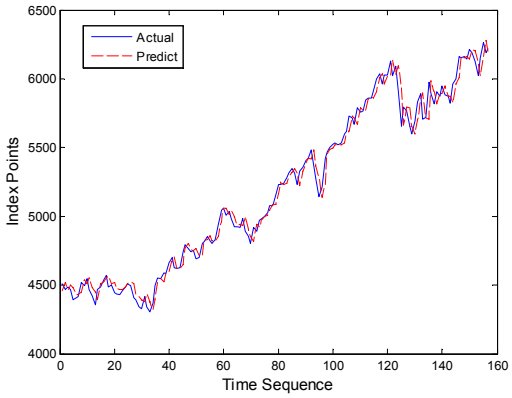


(f) NASDAQ testing data

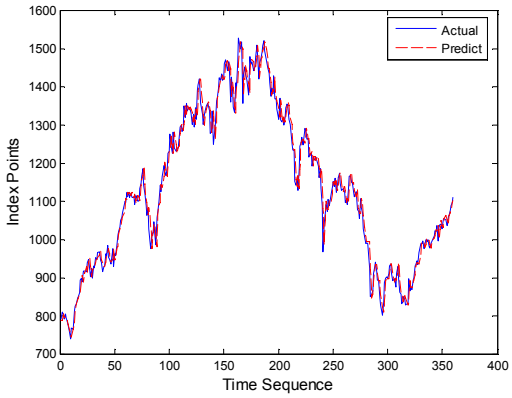
Figure 5.13: Training and testing data sets prediction - MORBT with FL = 50



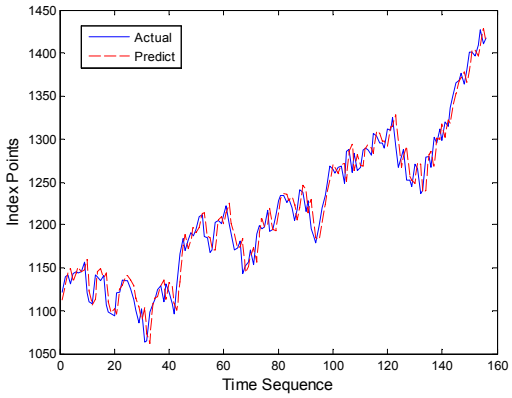
(a) FTSE training data



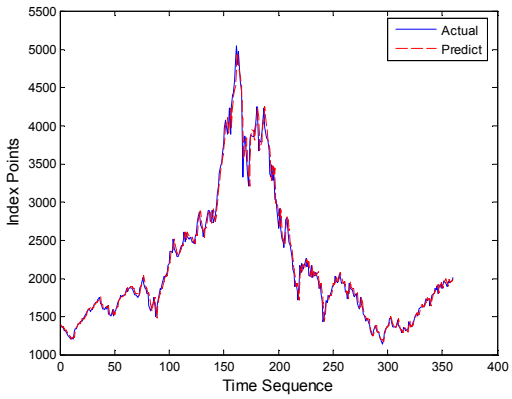
(b) FTSE testing data



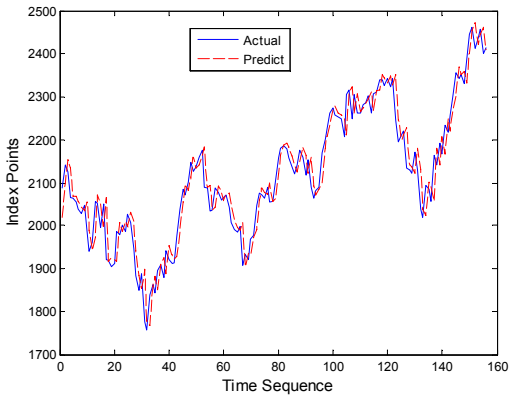
(c) S&P 500 training data



(d) S&P 500 testing data



(e) NASDAQ training data



(f) NASDAQ testing data

Figure 5.14: Training and testing data sets prediction - SP- MORBT with FL = 100

Table 5.8: Generalization error - MORBT with FL = 100

RMSE	MORBT (FL = 100)			
	Mean	Max	Min	SD
FTSE	74.0657	77.8578	72.6028	1.6240
S&P 500	17.0233	17.3281	16.8254	0.1573
NASDAQ	42.9483	43.6427	42.5009	0.4475

Table 5.9: Generalization error - MORBT with FL = 50

RMSE	MORBT (FL = 50)			
	Mean	Max	Min	SD
FTSE	74.7628	80.1057	72.8002	2.0480
S&P 500	17.0701	17.3031	16.8548	0.1572
NASDAQ	43.0834	44.2979	42.0798	0.5801

Results of the single phase algorithms are given in Table 5.10 and Table 5.11. The MORBT (FL=100) managed to obtain lower error than GA for all indexes. This shows that using different coefficients for the financial time series is important. The minimum RMSE are lower for MORBT across all data sets as compared to GA, however the maximum RMSE are higher. This translates to higher standard deviations of MORBT as compared to GA. This is most probably due to MORBT using different regression coefficient strings but in GA, only a single coefficient string is used. As compared to SP-MORBT, MORBT obtained lower mean errors for all the data sets.

Table 5.10: Generalization error - GA

RMSE	GA			
	Mean	Max	Min	SD
FTSE	75.1645	75.3337	74.9644	0.1098
S&P 500	17.1027	17.1365	17.0828	0.0195
NASDAQ	43.0546	43.0897	42.9867	0.0324

Table 5.11: Generalization error - SP-MORBT with FL = 100

RMSE	SP-MORBT (FL = 100)			
	Mean	Max	Min	SD
FTSE	74.9288	76.5860	72.7519	1.1113
S&P 500	17.1906	18.0985	16.8381	0.4053
NASDAQ	43.1199	44.2465	42.4851	0.5078

5.6 Conclusions

Different from existing approaches for TSF, a novel way of evolving rules to match regression coefficients to different zones of the time series in a multi-objective framework is proposed. Based on Pareto optimality, two conflicting objectives of minimizing error and maximizing rule coverage are considered. The minimization of error is required for good prediction while increasing coverage is mundane to cover a larger solution space. Experimental results show that the proposed approach is effective as a practical forecasting tool.

Chapter 6

Conclusions and Future Work

Automated intelligent systems are widely used in several industries for different applications as they are able to provide efficient solutions at lower cost in the long run. The uses of automated intelligent systems are once deemed only to replace human in manual work in order to allow experts to concentrate on more cognitive tasks. However, with the advance in artificial intelligence research, these systems are taking on a greater role in tasks that requires decision making. This thesis considered the shift in paradigm, presenting Computational Intelligence (CI) techniques for data analysis. As large amount of data is being collected, manual analysis would be inefficient. CI techniques for data analysis would be useful to complement and expedite human decisions.

Section 6.1 will conclude the work presented in this thesis. Section 6.2 will then suggest the possible future work and the general directions for this area of research.

6.1 Conclusions

This thesis has studied the use of CI techniques as automated systems for data analysis, in particular for classification and time series forecasting.

In Chapter 2, input features of classification problems are shown to exhibit certain degree of interference among each other. The Interference-Less Neural Network Training (ILNNT) method is proposed to reduce interference among input attributes by identifying those attributes that interfere with one another and separating them,

while attributes that are mutually beneficial are grouped together. Separated attributes in different batches do not share the same hidden neurons while attributes within a batch are connected to the same hidden neurons. With reduced interference, the networks are able to give better classification accuracy.

Chapter 3 then proposed a novel algorithm which uses EA with a newly formulated parameter, i.e., growth probability (P_g), to evolve the near optimal weights and the number of hidden neurons in NN. Training NN with growth probability-based evolution (NN-GP) initializes networks with only one hidden neuron and the networks are allowed to grow until a suitable size. Growth rate is based on Gaussian distribution thus providing a way to escape local optima. A self-adaptive version (NN-SAGP) with the aim of evolving the growth probability concurrently with NN during each generation is also proposed. The resultant networks are able to achieve high accuracy while using a small number of hidden neurons.

Subsequently in Chapter 4, a novel Evolutionary Memetic Algorithm (EMA) is proposed as a rule extraction mechanism. Two schemes for local search are studied, namely a Micro-Genetic Algorithm (μGA) local search, and an Artificial Immune System (AIS) inspired local search. EMA is complemented by the use of a variable length chromosome structure, which allows the flexibility to model the number of rules required.

Chapter 5 proposed a Multi-Objective Rule-Based Technique (MORBT) for time series forecasting. Dual phase is used in the algorithm, in phase I, the regression coefficients which are used to forecast the entire training output space are optimized. In phase II, rules are being evolved to localize the regression coefficients in a multi-objective framework. Phase II incorporates the concept of Pareto optimality to generate rules that are low in error and high in coverage. The staggered dual phase

approach adopted has the advantage of fragmenting the problem to promote better convergence, which is not achievable in a single phase due to the extremely large search space of such combinatorial optimization problem. The algorithm is applied on financial time series and has shown to produce good results. Therefore, Chapter 4 and Chapter 5 have shown that by having a good grasp of evolutionary rule techniques, the same fundamentals can be applied on different domains efficiently.

6.2 Future Work

Though several works considering different aspects of data analysis have been presented, however, this area is simply too vast to be fully encompassed in this thesis. The rest of this thesis would consider some possible future work to further improve on each of the proposed algorithm as well as to include an even more comprehensive research on computation intelligence techniques for data analysis.

The proposed future work for each chapter is first given in Section 6.2.1. Section 6.2.2 then states the general future directions of research work in this field that could be done.

6.2.1 Future Directions for Each Chapter

In Chapter 2, ILNNT could be applied to problems with any number of attributes, however the complexity scales up with the increase in the number of attributes due to the computation of interference table, as it requires C_2^N evaluations. For larger problems, a new and more efficient algorithm could be better though it is possible for the current algorithm to perform the task too. As this is a new area of research and there are plenty of possible avenues for improvements, its weakness can be remedied by some shortcut computation or distributed processing techniques. This could

include the use of divide-and-conquer techniques for the input space, etc. Hence, future work could be concentrated on formulating an efficient and faster algorithm for larger problems.

In Chapter 3, other types of evolutionary computation techniques could be used to evolve the NN architecture in a similar manner as proposed. It would be challenging and interesting to analyze and compare the different types of evolutionary computation techniques for this application.

Data in real life are often filled with noise and outliers. Proposing and incorporation of dedicated local search operators in Chapter 4 that are able to handle or filter out the noise and outliers would be extremely advantages.

In Chapter 5, the length of the sliding window could be a factor affecting the performance of the algorithm for time series forecasting. In the current MORBT version, the sliding window length is fixed throughout the evolution process, and hence varying this parameter might further improve the results. However, with the inclusion of sliding window length into the rules, the complexity and computational cost of the algorithm would inevitably increase. Therefore, the work would also include finding a solution that could incorporate an additional factor yet does not increase the cost significantly.

6.2.2 General Future Directions

As shown in Chapter 2, by using interference-less input space partitioning for NN, classification accuracy is improved. One possible direction of combining and complementing the different works in this thesis is to incorporate the partitioning within NN-GP/NN-SAGP presented in Chapter 3. This would again offer a myriad of possibilities for investigations. With the partitioning of the input space of NN-

GP/NN-SAGP, it is hoped that the performance of the network would be able to obtain higher generalization accuracy. In Chapter 3, the individual network evolved for each data set is near minimal, as the implementation of the partitioning would create a few sub-networks, this would pose a challenge of keeping the overall network size of NN-GP/NN-SAGP small.

The algorithms explored for classification has been restricted to static problems. In the real-world, there are frequent addition and deletion of features. The algorithms presented in this thesis could be modified as dynamic NN or dynamic evolutionary rule extraction algorithms where the number of input attributes or output classes can be subjected to changes. It would be meaningful and challenging to formulate and improve the algorithms to be suitable for such situations. The focus would be on modifying the existing algorithms and leverage upon what has been built. In order to form dynamic algorithms, it is inevitable to delete or add modules, e.g., sub-networks for NN or rules in the case of EA. The adaptability of the algorithms manifest in the consistent changes of its sub-modules and the difficulties would be in retaining the accuracy while making modifications.

Most of the algorithms presented in this thesis are used for classification, while only Chapter 5 shows application on time series forecasting. Future research on using CI techniques for time series forecasting could be carried out. Other aspects of time series forecasting could be studied.

Last but not least, more types of data analysis could be considered.

Bibliography

[1] H. A. Abbass, "An Evolutionary Artificial Neural Networks Approach for Breast Cancer Diagnosis," *Artificial Intelligence in Medicine*, vol. 25, pp. 265-281, 2002.

[2] H. A. Abbass, "Speeding Up Backpropagation using Multiobjective Evolutionary Algorithms", *Neural Computation*, vol. 15, issue 11, pp. 2705-2726, 2003.

[3] H. A. Abbass, M. Towsey and G. Finn, "C-net: A Method for Generating Non-Deterministic and Dynamic Multivariate Decision Trees", *Knowledge and Information Systems*, vol. 3, pp. 184–197, 2001.

[4] R. Agrawal, T. Imielinski and A. Swami, "Mining Association Rules between Sets of Items in Large Databases", in *Proceedings of SIGMOD Conference*, pp. 207-216, 1993.

[5] L. M. Almeida and T. B. Ludermir, "Tuning Artificial Neural Networks Parameters using an Evolutionary Algorithm", in *Proceedings of 8th International Conference on Hybrid Intelligent Systems*, pp. 927-930, 2008.

[6] R. T. Alves, M. R. Delgado, H. S. Lopes and A. A. Freitas, "An Artificial Immune System for Fuzzy-Rule Induction in Data Mining", in *Lecture Notes in Computer Science*, Berlin, Germany: Springer-Verlag, *Proceedings of Parallel Problem Solving From Nature*, vol. 3242, pp. 1011-1020, 2004.

[7] J. H. Ang, K. C. Tan and A. A. Mamun, "Training Neural Networks for Classification using Growth Probability-Based Evolution", *Neurocomputing*, vol. 71, pp. 3493–3508, October 2008.

[8] R. A. Araújo, A. R. L. Júnior and T. A. E. Ferreira, "A Quantum-Inspired Intelligent Hybrid Method for Stock Market Forecasting", in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1348-1355, 2008.

[9] J. Armstrong and F. Collopy, "Error Measures for Generalizing about Forecasting Methods - Empirical Comparisons", *International Journal of Forecasting*, vol. 8, no. 1, pp. 69-80, 1992.

[10] T. Ash, "Dynamic Node Creation in Backpropagation Networks" *Connection Science*, vol. 1, no. 4, pp. 365–375, 1989.

[11] F. Atienza, N. M. Alzamora, J. A. D. Velasco, S. Dreiseitl and L. O. Machado, "Risk Stratification in Heart Failure using Artificial Neural Networks", in *Proceedings of AMIA Symposium*, pp. 32-36, 2000.

[12] W. H. Au, K. C. C. Chan and A. K. C. Wong, "A Fuzzy Approach to Partitioning Continuous Attributes for Classification", *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 5, pp. 715-19, 2006.

[13] K. F. Au, T. M. Choi and Y. Yu, "Fashion Retail Forecasting by Evolutionary Neural Networks", *International Journal of Production Economics*, vol. 114, issue 2, pp. 615-630, 2008.

[14] S. Banik, F. H. Chanchary, R. A. Rouf and K. Khan, "Modeling Chaotic Behavior of Dhaka Stock Market Index Values using the Neuro-Fuzzy Model", in *Proceedings of 10th International Conference on Computer and Information Technology*, pp. 27-29, 2007.

[15] Y. Becerikli, "On Three Intelligent Systems: Dynamic Neural, Fuzzy and Wavelet Networks for Training Trajectory", *Neural Computing and Applications*, vol.13, no. 4, pp. 339-351, 2004.

[16] Y. Becerikli, Y, Oysal and A. F. Konar, "Trajectory Priming with Dynamic Fuzzy Networks in Nonlinear Optimal Control", *IEEE Transactions on Neural Networks*, vol.15, issue 2, pp.383-394, 2004.

[17] B. Blaha and D. Wunsch, "Evolutionary Programming to Optimize an Assembly Program", in *Proceedings of the IEEE Congress on Evolutionary Computation,* vol.2, pp 1901-1903, 2002.

[18] C. L. Blake and C. J. Merz, UCI Machine Learning Repository, Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[19] C. C. Bojarczuk, H. S. Lopes, A. A. Freitas, and E. L. Michalkiewicz, "A Constrained - Syntax Genetic Programming System for Discovering Classification Rules: Application to Medical Data Sets," *Artificial Intelligence in Medicine*, vol. 30, issue 1, pp. 27-48, 2004.

[20] B. L. Bowerman and R. T. O'Connell. Forecasting and Time Series: An Applied Approach. Belmont, California: Duxbury Press, 3rd Edition, 1993.

[21] G. Box, G. M. Jenkins and G. C. Reinsel. Time Series Analysis: Forecasting and Control. Prentice-Hall, 3rd edition. 1994.

[22] C. Brizuela, "Applications of Multi-Objective Evolutionary Algorithms", *IEEE Computational Intelligence Magazine*, vol. 1, issue 1, pp. 43-44, 2006.

[23] E. Cantu-Paz and C. Kamath, "An Empirical Comparison of Combinations of Evolutionary Algorithms and Neural Networks for Classification Problems", *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, vol. 35, pp. 915-927, 2005.

[24] D. R. Carvalho and A. A. Freitas, " A Hybrid Decision Tree/Genetic Algorithm for Coping with the Problem of Small Disjunct in Data Mining", in *Proceedings*

of Genetic and Evolutionary Computation Conference, Las Vegas, NV, USA, pp. 1061-1068, 2000.

[25] D. R. Carvalho and A. A. Freitas, "An Immunological Algorithm for Discovering Small - Disjunct Rules in Data Mining", in *Proceedings of Graduate Student Workshop at GECCO-2001*, San Francisco, USA, pp. 401-404, 2001.

[26] P. D. Castro, G. O. Coelho, M. F. Caetano and F. J. von Zuben, "Designing Ensembles of Fuzzy Classification Systems: An Immune-Inspired Approach", *in Proceedings of Fourth International Conference on Artificial Immune Systems* , vol. 3627, LNCS, pp. 469-482, 2005.

[27] H. Cen, Y. Bao and Y. He, "Pattern Recognition of Visible and Near-Infrared Spectroscopy from Bayberry Juice by use of Partial Least Squares and a Backpropagation Neural Network", *Applied Optics*, vol. 45, no. 29, pp. 7679-7683, 2006.

[28] N. H. Chan. Time Series: Applications to Finance. New York: Wiley-Interscience. 2002.

[29] M. S. Charifa, A. Suliman and M. Bikdash, "Face Recognition Using a Hybrid General Backpropagation Neural Network", in *Proceedings of the IEEE International Conference on Granular Computing*, pp. 510 – 515, 2007.

[30] T. H. Cheng, C. P. Wei and V.S. Tseng**,** "Feature Selection for Medical Data Mining: Comparisons of Expert Judgment and Automatic Approaches", in *Proceedings of 19th the IEEE International Symposium on Computer-Based Medical Systems*, pp. 165-170, 2006.

[31] S. C. Chiam, K. C. Tan and A. A. Mamun, "Multiobjective Evolutionary Neural Networks for Time Series Forecasting", *in Proceedings of Evolutionary Multi-Criterion Optimization*, pp. 346-360, 2007.

[32] S. C. Chiam, K. C. Tan and A. A. Mamum, "Evolutionary Multi-Objective Portfolio Optimization in Practical Context", *International Journal of Automation and Computing*, vol. 5, no. 1, pp. 67-80, 2008.

[33] C. A. Coello Coello, "Evolutionary Multi-Objective Optimization: A Historical View of the Field", *IEEE Computational Intelligence Magazine*, vol. 1, issue 1, pp. 28-36, 2006.

[34] C. A. Coello Coello and N. C. Cortes, "Solving Multi-Objective Optimization Problems using an Artificial Immune System", *Genetic Programming and Evolvable Machines*, vol. 6, no. 2, pp. 163-190, 2005.

[35] P. Cortez, M. Rocha and J. Neves, "A Meta-Genetic Algorithm for Time Series Forecasting", in *Proceedings of the 10th Portuguese Conference on Artificial Intelligence*, pp. 21-31, 2001.

[36] P. Cortez, M. Rocha and J. Neves, "Evolving Time Series Forecasting Neural Network Models", *in Proceedings of the 3rd International Symposium on Adaptive Systems: Evolutionary Computation and Probabilistic Graphical Models*, pp. 84-91, 2001.

[37] J. David, "The Performance of a Selection Architecture for Genetic Programming", in *Proceedings of the 11th European Conference on Genetic Programming,* LNCS, vol. 4971, p 170-181, 2008.

[38] L. N. de Castro and J. Timmis. Artificial Immune Systems: A New Computational Intelligence Approach. Berlin, Germany: Springer-Verlag. 2002.

[39] L. N. de Castro and F. J. V. Zuben, "Learning and Optimization Using the Clonal Selection Principle", *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 239-251, 2002.

[40] K. Deb, "Multi-objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems", *Evolutionary Computation*, vol. 7, no. 3, pp. 205-230, 1999.

[41] K. Deb. Multi-Objective Optimization Using Evolutionary Algorithms. New York: John Wiley & Sons. 2001.

[42] C. L. del Arco-Calderon, P. I. Vinuela and J. C. H. Castro, "Forecasting Time Series by Means of Evolutionary Algorithms", in *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature*, LNCS, vol. 3242, pp 1061-70, 2004.

[43] M. Doumpos and C. Zopounidis, "Multi–Criteria Classification Methods in Financial and Banking Decisions", *International Transactions in Operational Research*, vol. 9, issue 5, pp. 567 – 581, 2002.

[44] R. O. Duda and P. E. Hart. Pattern Classification and Scene Analysis. John Wiley & Sons. 1973.

[45] R. Eaton, J. Katupitiya, K. W. Siew and K. S. Dang, "Precision Guidance of Agricultural Tractors for Autonomous Farming", in *Proceedings of the IEEE Systems Conference*, pp. 1 – 8, 2008.

[46] S. E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture", *Advances in Neural Information Processing systems II*, D. S. Touretzky, San Mateo (ed), CA: Morgan Kaufmann, pp. 524–532, 1990.

[47] T. A. E. Ferreira, G. C. Vasconcelos and P. J. L. Adeodato, "A New Evolutionary Approach for Time Series Forecasting", in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining*, pp. 616-623, 2007.

[48] W. Finnoff, F. Hergert and H. G. Zimmermann, "Improving Model Selection by Non-Convergent Methods", *Neural Networks*, vol. 6, pp. 771-783, 1993.

[49] T. Fogarty, "Varying the Probability of Mutation in the Genetic Algorithm", in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 104-109, 1989.

[50] D. B. Fogel, E. C. Wasson and E. M. Boughton, "Evolving Neural Networks for Detecting Breast Cancer", *Cancer Letters*, vol. 96, no. 1, pp. 49–53, 1995.

[51] C. M. Fonseca and P. J. Fleming, "Genetic Algorithm for Multiobjective Optimization, Formulation, Discussion and Generalization", in *Proceeding of the Fifth International Conference on Genetic Algorithms*, pp. 416-423, 1993.

[52] E. Frank and I. H. Witten, "Generating Accurate Rule Sets without Global Optimization," in *Proceedings of the Fifteenth International Conference Machine Learning*, pp. 144-151, 1998.

[53] A. A. Freitas, "A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery', In: A. Ghosh and S. Tsutsui. (Eds.) Advances in Evolutionary Computation, Springer-Verlag, pp. 819-845. 2002.

[54] A. A. Freitas and J. Timmis, "Revisiting the Foundations of Artificial Immune Systems for Data Mining", *IEEE Transactions on Evolutionary Computation*, vol. 11, issue 4. pp. 521-540, 2007.

[55] A. L. Garcia-Almanza and E. P. K. Tsang, "Repository Method to Suit Different Investment Strategies", in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 790-797, 2007.

[56] S. Geman, E. Bienenstock and R. Doursat, "Neural Networks and the Bias/Variance Dilemma", *Neural Computation*, vol. 4, pp. 1-58, 1992.

[57] A. Glowacz and W. Glowacz, "Automatic Shape Recognition of Film Sequence with Application of Backpropagation Neural Network", in *Proceedings of the Conference on Human System Interactions*, pp. 76 – 81, 2008.

[58] C. K. Goh and K. C. Tan, "An Investigation on Noisy Environment in Evolutionary Multiobjective optimization", *IEEE Transactions on Evolutionary Computation*, vol. 11, issue 3, pp. 354-381, 2007.

[59] C. K. Goh, E. J. Teoh and K. C. Tan, " Hybrid Multiobjective Evolutionary Design for Artificial Neural Networks", *IEEE Transactions on Neural Networks*, vol. 19, no. 9, pp. 1531-1548, 2008.

[60] D. E. Goodman Jr, L. C. Boggess and A. B. Watkins, "Artificial Immune System Classification of Multiple-Class Problems", *Intelligent Engineering Systems through Artificial Neural Networks*, vol. 12, pp. 179-184, 2002.

[61] S. U. Guan and S. C. Li, "Incremental Learning with Respect to New Incoming Input Attributes", *Neural Processing Letters*, vol. 14, issue 3, pp. 241-260, 2001.

[62] S. U. Guan and P. Li, "A Hierarchical Incremental Learning Approach to Task Decomposition", *Journal of Intelligent Systems*, vol. 12, no. 3, pp. 201-226, 2002.

[63] S. U. Guan and P. Li, "Feature Selection for Modular Neural Network Classifiers", *Journal of Intelligent Systems*, vol. 12, no. 3, pp. 173-200, 2002.

[64] S. U. Guan and P. Li, "Incremental Learning in Terms of Output Attributes", *Journal of Intelligent Systems*, vol. 13, no. 2, pp. 95-122, 2004.

[65] S. U. Guan and J. Liu, "Incremental Ordered Neural Network Training", *Journal of Intelligent Systems*, vol. 12, no. 3, pp. 137-172, 2002.

[66] S. U. Guan and J. Liu, "Incremental Neural Network Training with an Increasing Input Dimension", *Journal of Intelligent Systems*, vol. 13, no. 1, pp. 43-69, 2004.

[67] S. U. Guan, J. Liu and Y. Qi, "An Incremental Approach to Contribution-Based Attributes Selection", *Journal of Intelligent Systems*, vol. 13, no. 1, pp. 15-42, 2004.

[68] G. Hackett and P. Luffrum. Business Decision Analysis: An Active Learning Approach. Oxford : Blackwell Business. 1999.

[69] M. T. Hagan and M. B. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm", *IEEE Transactions on Neural Networks*, vol. 5, issue 6, pp. 989-993, 1994.

[70] Y. Hayashi and R. Setiono, "Combining Neural Network Predictions for Medical Diagnosis", *Computers in Biology and Medicine*, vol. 3, issue 4, pp. 237-246, 2002.

[71] Y. Hayashi, R. Setiono and K. Yoshida, "A Comparison between Two Neural Network Rule Extraction Techniques for the Diagnosis of Hepatobiliary Disorders", *Artificial Intelligence in Medicine*, vol. 20, issue 3, pp. 205-216, 2000.

[72] S. Haykin. Neural Networks: A Comprehensive Foundations. Prentice Hall International, 2nd Edition. 1999.

[73] J. M. Herrero, X. Blasco, M. Martínez, C. Ramos and J. Sanchis, "Non-Linear Robust Identification of a Greenhouse Model using Multi-Objective Evolutionary Algorithms", *Biosystems Engineering*, vol. 98, issue 3, pp. 335-346, 2007.

[74] J. H. Holland. Adaptation in Natural and Artificial Systems. The MIT Press. 1992.

[75] A. R. Hoshmand. Business and Economic Forecasting for the Information Age: A Practical Approach. Westport, CT: Quorum Books. 2002.

[76] D. F. Hougen, M. Gini, and J. Slagle, "Partitioning Input Space for Reinforcement Learning for Control," in *Proceedings of the International Conference on Neural Networks*, vol. 2, pp. 755-760, 1997.

[77] Y. H. Hu and Y. L. Chen**, "**Mining Association Rules with Multiple Minimum Supports: A New Mining Algorithm and a Support Tuning Mechanism", *Decision Support Systems*, vol. 42, issue 1, pp. 1-24, 2006.

[78] H. T. Huynh and Y. Won, "Hematocrit Estimation from Compact Single Hidden Layer Feedforward Neural Networks Trained by Evolutionary Algorithm", in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2962-2966, 2008.

[79] H. Ishibuchi, T. Nakashima and T. Murata, "Comparison of the Michigan and Pittsburgh Approaches to the Design of Fuzzy Classification Systems", *Electronics & Communications in Japan, Part III: Fundamental Electronic Science (English translation of Denshi Tsushin Gakkai Ronbunshi)*, vol. 80, no. 12, pp. 10-19, 1997.

[80] G. Janacek. Practical Time Series. London: Arnold; New York: Oxford University Press. 2001.

[81] A. Jaszkiewicz, "Do Multiple-Objective Metaheuristics Deliver on their Promises? A Computational Experiment on the Set-Covering Problem", *IEEE Transactions on Evolutionary Computation*, vol. 7, issue 2, pp. 133-143, 2003.

[82] A. R. L. Júnior, T. A. E. Ferreira and R. de A. Araújo, "An Experimental Study with a Hybrid Method for Tuning Neural Network for Time Series Prediction", in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 3434-3441, 2008.

[83] H. Kahramanli and N. Allahverdi, "Rule Extraction from Trained Adaptive Neural Networks using Artificial Immune Systems", *Expert Systems with Applications*, vol. 36, issue 2, part 1, pp.1513-1522, 2009.

[84] N. B. Karayiannis and M. M. Randolph-Gips, "On the Construction and Training of Reformulated Radial Basis Function Neural Networks", *IEEE Transactions on Neural Networks*, vol. 14, issue 4, pp. 835 – 846, 2003.

[85] T. Kerstetter, S. Massey and J. Roberts, "Recursively Partitioning Neural Networks for Radar Target Recognition," in *Proceedings of International Joint Conference on Neural Networks*, vol. 5, pp. 3208-3212, 1999.

[86] P. S. Knopov and T. V. Pepelyaeva, "Some Trading Strategies on the Securities Market", *Cybernetics and Systems Analysis*, vol. 38, no. 5, pp. 736-739, 2002.

[87] J. D. Knowles and D. W. Corne, "M-PAES: A Memetic Algorithm for Multiobjective Optimization", *in Proceedings of the Congress on Evolutionary Computation*, vol. 1, pp. 325-332, 2000.

[88] M. Kobayashi, A. Zamani, S. Ozawa and S. Abe, "Reducing Computations in Incremental Learning for Feedforward Neural Network with Long-Term Memory", in *Proceedings of the IEEE International Joint Conference on Neural Networks*, vol 3. pp. 1989-1994, 2001.

[89] T. Kohonen, "The Self-Organizing Map", in *Proceedings of the IEEE*, vol. 78, issue 9, pp. 1464 – 1480, 1990.

[90] A. Konstantaras, M. R. Varley, F. Vallianatos, G. Collins and P. Holifield, "A Neuro-Fuzzy Approach to the Reliable Recognition of Electric Earthquake Precursors", *Natural Hazards and Earth Systems Science*, vol. 4, pp. 641-646, 2004.

[91] K. Krawiec, "Generative Learning of Visual Concepts using Multiobjective Genetic Programming", *Pattern Recognition Letters*, vol. 28, issue 16, pp. 2385-2400, 2007.

[92] K. J. Lang, A. H. Waibel and G. E. Hinton, "A Time-Delay Neural Network Architecture for Isolated Word Recognition" *Neural Networks*, vol. 3, no. 1, pp. 33-43, 1990.

[93] M. Lehtokangas, "Modelling with Constructive Backpropagation," *Neural Networks*, vol. 12, pp. 707-716, 1999.

[94] L. Lepistö, I. Kunttu and A. Visa, "Rock Image Classification based on *k*-Nearest Neighbour Voting", in *Proceedings of the IEE Vision, Image, and Signal Processing*, vol. 153, issue 4, p. 475-482, 2006.

[95] F. H. F. Leung, H. K. Lam, S. H. Ling and P. K. S. Tam, "Tuning of the Structure and Parameters of a Neural Network using an Improved Genetic Algorithm," *IEEE Transactions on Neural Networks*, vol 14, issue 1, pp. 79 – 88, 2003.

[96] N. S. Lewis, M. Pardo, B. C. Sisk and G. Sberveglieri, "Comparison of Fisher's Linear Discriminant to Multilayer Perceptron Networks in the Classification of Vapors using Sensor Array Data", *Sensors and Actuators B (Chemical)*, vol. 115, no. 2, pp. 647-655, 2006.

[97] M. A. Lifshits**.** Gaussian Random Functions. Dordrecht, Boston: Kluwer Academic Publishers. 1995.

[98] Y. P. Lin, C. H. Wang, T. L. Wu, S. K. Jeng and J. H. Chen, "Multilayer Perceptron for EEG Signal Classification during Listening to Emotional Music", in *Proceedings of the IEEE Region 10 Conference*, pp. 1-3, 2007.

[99] J. K. Lindsey. Introductory Statistics: A Modelling Approach. Oxford, New York: Clarendon Press. 1995.

[100] Y. Liu and X. Yao, "Evolving Neural Networks for Hang Seng Stock Index Forecast", in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 1, pp. 256-260, 2001.

[101] G-C. Luh and C-H. Chueh, "Multi-Objective Optimal Design of Truss Structure with Immune Algorithm", *Computers and Structures*, vol. 82, issue 11-12, pp. 829-844, 2004.

[102] G. C. Luh, C. H. Chueh and W. W. Liu, "MOIA: Multi-Objective Immune Algorithm", *Engineering Optimization*, vol. 35, no. 2, pp. 143-164, 2003.

[103] C. Luque, J. M. V. Ferran and P. I. Vinuela, "Time Series Forecasting by Means of Evolutionary Algorithms", in *Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pp.1 – 7, 2007.

[104] L. Ma and K. Khorasani "New Training Strategies for Constructive Neural Networks with Application to Regression Problems," *Neural Networks*, vol. 17, no. 4, pp. 589-609, 2004.

[105] L. Ma and K. Khorasani, "Constructive Feedforward Neural Networks using Hermite Polynomial Activation Functions," *IEEE Transaction on Neural Networks*, vol. 16, no. 4, pp. 821-833, 2005.

[106] M. Mahfouf, M. F. Abbod and D. A. Linkens, "A Survey of Fuzzy Logic Monitoring and Control Utilisation in Medicine," *Artificial Intelligence in Medicine*, vol. 12, issue 1, pp. 27-42, 2001.

[107] S. G. Makridakis and S. C. Wheelwright. Forecasting Methods and Applications. New York: Wesley. 1998.

[108] O. L. Mangasarian and W. H. Wolberg, "Cancer Diagnosis via Linear Programming", *SIAM News*, vol. 23, no. 5, pp. 1-18, 1990.

[109] U. Markowska-Kaczmar and W. Trelak, "Fuzzy Logic and Evolutionary Algorithm—Two Techniques in Rule Extraction from Neural Networks", *Neurocomputing*, vol. 63, pp. 359-379, 2005.

[110] H. A. Mayer and R. Schwaiger, "Evolutionary and Coevolutionary Approaches to Time Series Prediction using Generalized Multi-Layer Perceptrons", in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 1, pp. 6-9, 1999.

[111] P. Merz and B. Freisleben, "A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem", in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 1, pp. 2063-2070, 1999.

[112] T. P. Meyer and N. H. Packard, "Local Forecasting of High-Dimensional Chaotic Dynamics", *Nonlinear Modeling and Forecasting*, M. Casdagli, and S. Eubank (Eds), pp. 249-263, 1990.

[113] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. London: Kluwer Academic Publisher. 1994.

[114] D. Michie, D. Spiegelhalter and C. Taylor. Machine Learning, Neural and Statistical Classification. Hemel Hempstead, Hertfordshire, England: Ellis Horwood. 1994.

[115] S. Mitra, S. K. Pal and P. Mitra, "Data Mining in Soft Computing Framework: A Survey", *IEEE Transactions on Neural Networks*, vol. 13, issue 1, pp. 3-13, 2002.

[116] D. C. Montgomery, G. C. Runger and N. F. Hubele. Engineering Statistics. New York : Wiley, John & Sons, Inc., 2nd ed. 2001.

[117] N. Morgan and H. Bourlard, "Generalization and Parameter Estimation in Feedforward Nets: Some Experiments" *Advances in Neural Information Processing Systems*, pp. 630-637, 1990.

[118] P. Moscato, "Memetic Algorithms: A Short Introduction", In D. Corne, M. Dorigo and F Glover (Eds), *New Ideas in Optimization,* pp. 219-234. McGraw-Hill, 1999.

[119] R. F. Mould. Introductory Medical Statistics. Bristol, Philadelphia, Institute of Physics Pub., 3rd edition. 1998.

[120] K. A. Nagaty, "Fingerprints Classification using Artificial Neural Networks: A Combined Structural and Statistical Approach", *Neural Networks*, vol. 14, no. 9, pp. 1293-305, 2001.

[121] P. S. Ngan, M. L. Wong, W. Lam, K. S. Leung and C. Y. Cheng, "Medical Data Mining using Evolutionary Computation," *Artificial Intelligence in Medicine*, vol. 16, issue 1, pp. 73-96, 1999.

[122] E. Noda, A. A. Freitas and H. S. Lopes, "Discovering Interesting Prediction Rules with a Genetic Algorithm", in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1322-1329, 1999.

[123] S. N. Omkar, R. Khandelwal, S. Yathindra, G. N. Naik and S. Gopalakrishnan, "Artificial Immune System for Multi-Objective Design Optimization of Composite Structures", *Engineering Applications of Artificial Intelligence*, vol.21, issue 8, pp. 1416-1429, 2008.

[124] Y. S, Ong and A. J. Keane, "Meta-Lamarckian Learning in Memetic Algorithms", *IEEE Transactions on Evolutionary Computation*, vol. 8, issue 2, pp. 99-110, 2004.

[125] P. P. Palmes, T. Hayasaka and S. Usui, "Mutation-Based Genetic Neural Network," *IEEE Transactions on Neural Networks*, vol. 16, issue 3, pp. 587-600, 2005.

[126] K. N. Pantazopoulos, L. H. Tsoukalas, N. G. Bourbakis, M. J. Brun and E. N. Houstis, "Financial Prediction and Trading Strategies using Neurofuzzy Approaches", *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 28, issue 4, pp. 520 – 531, 1998.

[127] J. Park and D. W. Edington, "A Sequential Neural Network Model for Diabetes Prediction", *Artificial Intelligence in Medicine*, vol. 23, issue 3, pp. 277-293, 2001.

[128] D. Plikynas, "Decision Rules Extraction from Neural Network: A Modified Pedagogical Approach," *Informacines Technologijos IR Valdymas*, vol. 31, no. 2, pp. 53-59, 2004.

[129] M. Pourahmadi. Foundations of Time Series Analysis and Prediction Theory. New York; Singapore: John Wiley & Sons. 2001.

[130] L. Prechelt, "PROBEN1: A Set of Neural Network Benchmark Problems and Benchmarking Rules," Technical Report 21/94, University of Karlsruhe, Germany, Department of Informatics, 1994.

[131] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling. Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press. 1992.

[132] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[133] J. Rajan and V Saravanan, "A Framework of an Automated Data Mining System Using Autonomous Intelligent Agents", in *Proceedings of the International Conference on Computer Science and Information Technology*, pp. 700-704, 2008.

[134] D. G. Rees. Essential Statistics for Medical Practice: A Case-Study Approach. London, New York: Chapman and Hall. 1994.

[135] M. Remzi and B. Djavan, "Artificial Neural Networks in Urology", *European Urology Supplements*, vol. 3, issue 3, pp. 33-38, 2004.

[136] M. Rimer and T. Martinez, "CB3: An Adaptive Error Function for Backpropagation Training", *Neural Processing Letters*, vol. 24, no. 1, pp. 81-92, 2006.

[137] E. Romero and J. M. Sopena, "Performing Feature Selection with Multilayer Perceptrons", *IEEE Transactions on Neural Networks*, vol. 19, issue 3, p 431-441, 2008.

[138] F. Rovira-Mas, S. Han and J. F. Reid, "Evaluation of Automatically Steered Agricultural Vehicles", in *Proceedings of the IEEE Symposium on Position, Location and Navigation,* pp. 473-478, 2008.

[139] A. Savasere, E. Omiecinski and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases." in *Proceedings of the 21st International Conference on Very Large Data Bases*, pp. 432-444, 1995.

[140] R. Setiono, W. K. Leow and J. M. Zurada**, "**Extraction of Rules from Artificial Neural Networks for Nonlinear Regression" *IEEE Transactions on Neural Networks*, vol. 13, issue 3, pp. 564-577, 2002.

[141] Y. Shao, J. Li, J. Wang, T. Chen, F. Tian and J. Wang, "Model and Simulation of Stock Market based on Agent", in *Proceedings of the International Conference on Information and Automation*, pp. 248 – 252, 2008.

[142] G. Shi and Q. Ren, "Research on Compact Genetic Algorithm in Continuous Domain", in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 793-800, 2008.

[143] A. Shilton, M. Palaniswami, D. Ralph and A. C. Tsoi, "Incremental Training of Support Vector Machines", *IEEE Transactions on Neural Networks*, vol. 16, issue 1, pp. 114-131, 2005.

[144] J. Smith and T. C. Fogarty, "Self Adaptation of Mutation Rates in a Steady State Genetic Algorithm", in *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 318 – 323, 1996.

[145] N. Srinivas and K. Deb, "Multiobjective Optimization using Non-Dominated Sorting in Genetic Algorithms", *Evolutionary Computation*, vol. 2, no. 3, pp. 221-248, 1994.

[146] D. Stathakis and A. Vasilakos, "Satellite Image Classification using Granular Neural Networks", *International Journal of Remote Sensing*, vol. 27, no. 18, pp. 3991- 4003, 2006.

[147] L. Su, S. U. Guan and Y. C. Yeo, "Incremental Self-Growing Neural Networks with the Changing Environment", *Journal of Intelligent Systems*, vol. 11, no. 1, pp. 43-74, 2001.

[148] R. Sun and T. Peterson, "Multi-Agent Reinforcement Learning: Weighting and Partitioning", *Neural Networks*, vol. 12, pp. 727-753, 1999.

[149] T. H. Sun and F. C. Tien, "Using Backpropagation Neural Network for Face Recognition with 2D + 3D Hybrid Information", *Expert Systems with Applications*, vol. 35, no. 1-2, pp. 361-372, 2008.

[150] A. A. Suratgar, M. B. Tavakoli and A. Hoseinabadi "Modified Levenberg-Marquardt Method for Neural Networks Training", *Transactions on Engineering, Computing and Technology*, vol. 6, pp. 46-48, 2005.

[151] N. Svangard, P. Nordin, S. Lloyd and C. Wihlborg, "Evolving Short-Term Trading Strategies using Genetic Programming", in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 2, pp. 2006 – 2010, 2002.

[152] H. Talib and J. M. Saleh, "Performance Comparison of Various MLPs for Material Recognition based on Sonar Data", in *Proceedings of the International Symposium on Information Technology*, vol. 4, pp. 1-4, 2008.

[153] K. C. Tan, Y. H. Chew and T. H. Lee, "A Hybrid Multi-Objective Evolutionary Algorithm for Solving Vehicle Routing Problem with Time Windows", *Computational Optimization and Applications*, vol. 34, pp. 115-151, 2006.

[154] K. C. Tan, C. K. Goh, A. A. Mamun and E. Z. Ei, "An Evolutionary Artificial Immune System for Multi-Objective Optimization", *European Journal of Operation Research*, issue 187, pp. 371-392, 2008.

[155] K. C. Tan, C. K. Goh, Y. J. Yang and T. H. Lee, "Evolving Better Population Distribution and Exploration in Evolutionary Multi-Objective Optimization", *European Journal of Operational Research*, issue 171, pp. 463-495, 2006.

[156] K. C. Tan, E. F. Khor, T. H. Lee and R. Sathikannan, "An Evolutionary Algorithm with Advanced Goal and Priority Specificiation for Mulit-objective Optimization", *Journal of Artificial Intelligence Research*, vol. 18, pp. 183-215, 2003.

[157] K.C. Tan, T. H. Lee and E. F. Khor, "Evolutionary Algorithms with Dynamic Population Size and Local Exploration for Multiobjective Optimization", *IEEE Transaction on Evolutionary Computation*, vol. 5, no. 6, pp. 565-588, 2001.

[158] K. C. Tan, T. H. Lee and E. F. Khor, "Evolutionary Algorithms for Multi-objective Optimization: Performance Assessments and Comparisons", *Artificial Intelligence Review*, vol. 17, pp. 253-290, 2002.

[159] K. C. Tan, A. Tay, T. H. Lee and C. M. Heng, "Mining Multiple Comprehensible Classification Rules using Genetic Programming", *in Proceeding of* the *IEEE International Congress on Evolutionary Computation,* Honolulu, Hawaii, USA, pp. 1302-1307, 2002.

[160] K. C. Tan, Q. Yu and J. H. Ang, "A Coevolutionary Algorithm for Rules Discovery in Data Mining", *International Journal of Systems Science*, vol. 37, no. 12, pp. 835-864, 2006.

[161] K. C. Tan, Q. Yu and J. H. Ang, "A Dual-Objective Evolutionary Algorithm for Rules Extraction in Data Mining", *Computational Optimization and Applications*, vol. 34, pp. 115-151, 2006.

[162] K. C. Tan, Q. Yu, C. M. Heng and T. H. Lee, "Evolutionary Computing for Knowledge Discovery in Medical Diagnosis", *Artificial Intelligence in Medicine*, vol. 27, pp. 129-154, 2003.

[163] R. Tavakkoli-Moghaddam, N. Safaei and F. Sassani, "A Memetic Algorithm for the Flexible Flow Line Scheduling Problem with Processor Blocking", *Computers & Operations Research*, vol.36, issue 2, pp. 402-414, 2009.

[164] D. Thierens, "Adaptive Mutation Rate Control Schemes in Genetic Algorithms", in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 1, pp. 980 – 985, 2002.

[165] J. Timmis, "Artificial Immune Systems - Today and Tomorrow", *Natural Computing*, vol. 6, no. 1, pp. 1-18, 2007.

[166] J. Timmis, A. Hone, T. Stibor and E. Clark, "Theoretical Advances in Artificial Immune Systems", *Theoretical Computer Science*, vol. 403, no. 1, pp. 11-32, 2008.

[167] J. Timmis and M. Neal, "A Resource Limited Artificial Immune System for Data Analysis", *Knowledge-Based Systems*, vol. 14, pp. 121-130, 2001.

[168] J. Timmis, M. Neal and J. Hunt, "An Artificial Immune System for Data Analysis", *BioSystems*, vol. 55, pp. 413-150, 2000.

[169] R. Tinós and A. C. P. L. F. de Carvalho, "Use of Gene Dependent Mutation Probability in Evolutionary Neural Networks for Non-Stationary Problems", *Neurocomputing*, vol. 70, issue 1-3, pp. 44-54, 2006.

[170] G. G. Towell and J. W. Shavlik, "Knowledge-based Artificial Neural Networks", *Artificial Intelligence*, vol. 70, issues 1-2, pp. 119-165, 1994.

[171] V. S. Tseng, C. H. Chen, P. C. Huang and T. P. Hong, "A Cluster-based Genetic Approach for Segmentation of Time Series and Pattern Discovery", in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1949-1953, 2008.

[172] H. Tsukimoto, "Extracting Rules from Trained Neural Networks", *IEEE Transactions on Neural Networks*, vol. 11, issue 2 , pp. 377-389, 2000.

[173] E. D. Ubeyli and I. Guler, "Improving Medical Diagnostic Accuracy of Ultrasound Doppler Signals by Combining Neural Network Models", *Computers in Biology and Medicine*, vol. 35, issue 6, pp. 533-554, 2005.

[174] J. A. Vasconcelos, J. A. Ramirez, R. H. C. Takahashi and R. R. Saldanha, "Improvements in Genetic Algorithms", *IEEE Transactions on Magnetics*, vol. 37, issue 5, part 1, pp. 3414 – 3417, 2001.

[175] M. M. B. R. Vellasco, M. A. C. Pacheco, L. S. R. Neto and F. J. D. Souza, "Electric Load Forecasting: Evaluating the Novel Hierarchical Neuro-Fuzzy BSP Model", *International Journal of Electrical Power and Energy Systems*, vol. 26, pp. 131-142, 2004.

[176] B. Verma and R. Ghosh, "A Novel Evolutionary Neural Learning Algorithm", in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1884 – 1889, 2002.

[177] G. G. Vining. Statistical Methods for Engineers. Pacific Grove, California: Duxbury Press. 1997.

[178] C. Wang, X. Wang and X. Zhang, "Research on the Improved Frequent Predicate Algorithm in the Data Mining of Criminal Cases", in *Proceedings of the International Conference on Information and Automation*, pp. 1531 – 1535, 2008.

[179] A. B. Watkins and L. C. Boggress, "A Resource Limited Artificial Immune Classifier", in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 1, 12-17, pp. 926 – 931, 2002.

[180] A. B. Watkins and L. C. Boggress, "A New Classifier based on Resource Limited Artificial Immune Systems", in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol.2, pp. 1546-1551, 2002.

[181] S. Weaver, L. Baird and M. Polycarpou, "Using Localizing Learning to Improve Supervised Learning Algorithms", *IEEE Transactions on Neural Networks*, vol. 12, pp. 1037-1046, 2001.

[182] J. M. Wei, W. G. Yi and M. Y. Wang, "Novel Measurement for Mining Effective Association Rules", *Knowledge-Based Systems*, vol. 19, issue 8, pp.739-743, 2006.

[183] T. Weise, M. Zapf and K. Geihs, "Rule-based Genetic Programming", in *Proceedings of the Bio-Inspired Models of Network, Information, and Computing Systems, Bionetics*, pp. 8-15, 2007.

[184] Yahoo! Finance Website.

[185] W. Yan, S. Lu and D. C. Yu, "A Novel Optimal Reactive Power Dispatch Method based on an Improved Hybrid Evolutionary Programming Technique", *IEEE Transactions on Power Systems*, vol. 19, issue 2, pp. 913-918, 2004.

[186] J. M. Yang and C. Y. Kao, "A Robust Evolutionary Algorithm for Training Neural Networks", *Neural Computing & Applications*, vol. 10, no. 3, pp. 214-230, 2001.

[187] J. H. Yang, L. Sun, H. P. Lee, Y. Qian and Y. C. Liang, "Clonal Selection Based Memetic Algorithm for Job Shop Scheduling Problems", *Journal of Bionic Engineering*, vol. 5, issue 2, pp. 111-119, 2008.

[188] Q. Yang, J. Yin, C. Ling and R. Pan, "Extracting Actionable Knowledge from Decision Trees", *IEEE transactions on Knowledge and Data Engineering*, vol. 19, issue 1, pp. 43-56, 2007.

[189] X. Yao and Y. Liu, "Neural Networks for Breast Cancer Diagnosis", in *Proceedings of the IEEE Congress on Evolutionary Computation,* vol. 3, pp. 1760-1767, 1999.

[190] E. Zio, P. Baraldi and N. Pedroni, "Optimal Power System Generation Scheduling by Multi-Objective Genetic Algorithms with Preferences", *Reliability Engineering and System Safety*, vol. 94, pp. 432– 444, 2009.

[191] O. Zoran and S. Rangarajan, "Constructive Neural Networks Design using Genetic Optimization," *Series Mathematics and Informatics*, vol. 15, pp. 133-146, 2000.