Mining continuous classes using evolutionary computing

by

Gavin Potgieter

Submitted in partial fulfilment of the requirements for the degree Magister Scientiae

in the Faculty of Natural and Agricultural Science

University of Pretoria

Pretoria

October 2002

Mining continuous classes using evolutionary computing

by

Gavin Potgieter

Abstract

Data mining is the term given to knowledge discovery paradigms that attempt to infer knowledge, in the form of rules, from structured data using machine learning algorithms. Specifically, data mining attempts to infer rules that are accurate, crisp, comprehensible and interesting. There are not many data mining algorithms for mining continuous classes. This thesis develops a new approach for mining continuous classes. The approach is based on a genetic program, which utilises an efficient genetic algorithm approach to evolve the non-linear regressions described by the leaf nodes of individuals in the genetic program's population. The approach also optimises the learning process by using an efficient, fast data clustering algorithm to reduce the training pattern search space. Experimental results from both algorithms are compared with results obtained from a neural network. The experimental results of the genetic program is also compared against a commercial data mining package (Cubist). These results indicate that the genetic algorithm technique is substantially faster than the neural network, and produces comparable accuracy. The genetic program produces substantially less complex rules than that of both the neural network and Cubist.

Thesis supervisor: Prof. A. P. Engelbrecht
Department of Computer Science
Degree: Magister Scientiae

Acknowledgements

I would like to thank the following people for their assistance during the production of this thesis:

- Professor A.P. Engelbrecht, my thesis supervisor, for his insight and motivation;
- Andrew du Toit, Andrew Cooks and Jacques van Greunen, UP Techteam members, for maintaining the computer infrastructure used to perform my research;
- Frans van den Bergh and Edwin Peer who listened patiently when I discussed some of my ideas with them, for their feedback and insight.

Felix qui potuit rerum cognoscere causas

Happy is he who has been able to understand the cause of things

— Virgil

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Knowledge discovery is the process of obtaining useful knowledge from raw data or facts. Knowledge can be inferred from data by a computer using a variety of *machine learning* paradigms. Data mining is the generic term given to knowledge discovery paradigms that attempt to infer knowledge in the form of rules from structured data using machine learning.

In an article in the Boston Sunday Globe of August 11 2002, Vest identified the Biotechnology industry as the largest growing industry in Boston [97]. This follows the successful conclusion of the human genome project. The result of the human genome project is essentially a catalogue and guide to the structure of human DNA. Yet, very little is known about the effects of individual genes in DNA. In order to decipher and analyse the vast amount of information stored in DNA, data mining will have to be performed on a massive scale.

As an industry, data mining and data warehousing threatens to replace e-commerce as the major information technology driving force of the 21st century. Recent events in the media also illustrate the need for data mining:

- The September 11 2001 attacks on the World Trade Centre in New York highlighted major deficiencies in the method of information gathering used by US agencies such as the FBI and the CIA. Data mining could be used as an effective tool to combat criminal and terrorist activities, by tracing the activities, communications and purchases of individuals and identifying possible suspects.

- The recent scandals involving CEOs of large multi-national corporations such as Enron, Worldcom, *etc.*, illustrate the need for the development of data mining methods in order

to detect the fraudulent activities of individuals and groups of individuals.

- The recent world-wide recession and collapse of the airline industry illustrate the need for companies to improve their market share as a means of survival. Access to accurate and reliable information is crucial for decision making. Data mining could be used to provide a competitive edge over other companies.

Thus, knowledge discovery in the form of data mining is becoming increasingly important in today's world. However, in order to perform some of the mammoth tasks described above, it is necessary to develop fast, efficient knowledge discovery algorithms.

Knowledge discovery algorithms can be divided into two main categories according to their learning strategies:

- **Supervised** learning algorithms attempt to minimise the error between their predicted outputs and the target outputs of a given dataset. The target outputs can either be
  - *discrete*, *i.e.* the supervised learning algorithm attempts to predict the class of a problem, *e.g.* whether it will be sunny, rainy or overcast in tomorrow's forecast,
  - or *continuous*, *i.e.* the supervised learning algorithm attempts to predict the value associated with a class, *e.g.* determining the price of a VCR.
- **Unsupervised** learning algorithms attempt to cluster a dataset into homogeneous regions, according to some characteristic present in the data.

Many knowledge discovery algorithms have been developed which utilise machine learning and artificial intelligence paradigms. The main classes of paradigms include: artificial neural networks [13][106], classification systems (like ID3 [81], CN2 [20][21] and C4.5 [83]), evolutionary computing [7][8][49], regression systems (like M5 [82]) *etc.*

One of the primary problems with current data mining algorithms is the scaling of these algorithms for use on large databases. Additionally, very little attention has been paid to algorithms for mining with continuous target outputs, which requires non-linear regression. This thesis presents and fully discusses a number of evolutionary computing algorithms suitable for non-linear regression. The primary algorithm developed by this thesis is a genetic program for the mining of continuous classes (GPMCC), which utilises a genetic algorithm that evolves

structurally optimal polynomial expressions (GASOPE) as the non-linear regressions for the terminal nodes of the individuals in the genetic program. Additionally, a number of algorithms are developed which optimise the learning process. These algorithms utilise a fast, efficient data clustering algorithm. Methods for coping with large databases are also covered by this thesis.

The remainder of this thesis is organised as follows: Chapter 2 provides a taxonomy of data mining methods in use today. It also covers many of the principles used throughout this thesis. The GASOPE method is presented in chapter 3 and details other methods employed by or related to the GASOPE method. Chapter 4 presents the GPMCC method, fully discusses the GPMCC method's interaction with the GASOPE method and provides an overview of other associated methods. Finally, chapter 5 concludes this thesis

Chapter 2

BACKGROUND

This chapter provides a taxonomy of the methods employed by many data mining applications in use today. An overview of the methods utilised in later sections is also presented. The paradigms of knowledge discovery, evolutionary computing, neural networks and clustering are discussed in broad terms. Attention to detail is given for specific methods as and when they are required by later chapters.

2.1 KNOWLEDGE DISCOVERY

This section discusses knowledge discovery and, more specifically, knowledge discovery through machine learning. Section 2.1.1 introduces the reader to the fundamentals of knowledge discovery. Some of the more common types of rule induction algorithms are discussed in section 2.1.2. Finally, section 2.1.3 provides examples of the past usage of various types of rule induction algorithms and is not confined to the induction algorithms presented in section 2.1.2.

2.1.1 Introduction

Knowledge discovery is the process of obtaining useful knowledge from raw data or facts. Such data can be structured, *e.g.* relational databases, tables and spreadsheets, or unstructured, *e.g.* text, video and audio. Knowledge discovery applied to structured data is termed *data mining* and knowledge discovery applied to unstructured data is usually termed *text mining*.

Knowledge can be defined, for the purpose of this thesis, as an organised body of information. Such an organised body of information can be represented by rules of the form

IF *antecedent* THEN *consequent*

where the antecedent describes a test on the state of a set of attributes and the consequent describes a response to that state. The goal of knowledge discovery is thus to infer rules from data that are

- **accurate**,
- **comprehensible**,
- **crisp**,
- and **interesting**.

One way in which knowledge can be inferred from data is through *machine learning*, by an algorithmic process known as *empirical learning*. Empirical learning includes algorithms that reason from supplied examples in order to produce general theories about a specific problem domain, which is categorised by a dataset. These general theories are used to make predictions about further points in the problem domain, through both extrapolation and interpolation. Examples of empirical learning algorithms include *artificial neural networks* (discussed in section 2.3), *evolutionary computing* (discussed in section 2.2) and the large variety of rule-induction algorithms presented later in this section.

If training examples are supplied with known labels or targets, the empirical learning strategy is called *supervised learning*. Otherwise, the empirical learning strategy is known as *unsupervised learning*. Supervised learning can solve two types of problems: *classification* problems, where the training example labels are categorical and are called *classes*, and *regression* problems, where the training example labels are continuous.

2.1.2 Definitions

This section provides definitions of terms used throughout this thesis. A number of different rule induction algorithms are also discussed. The rule induction algorithms are divided into two categories according to the types of problems they solve:

- **Classification systems**, which classify training examples according to their discrete target outputs.
- **Regression systems**, which predict the numeric value associated with a class, *i.e.* regression systems predict the continuous target outputs of training examples.

Classification systems

A *decision tree* is one representation of an organised body of information. A decision tree can recursively be defined as either a leaf node (terminal node) that names a class, or as an internal node (non-terminal node) that represents an attribute-based test, with a branch to another decision tree for each outcome of that test. A node that is linked to a node higher up in the hierarchy is called a *child* node. The sequence of nodes from the *root* of the tree to a leaf node is called a *path*. All paths of a decision tree are mutually exclusive and exhaustive, *i.e.* all regions defined by all paths completely cover the *instance space* (or pattern space). An instance space is an n-dimensional attribute space where each instance describes a point in that space. The *size* of a decision tree is the number of nodes in it, including the leaf nodes.

Each attribute in a decision tree can either be nominal, ordinal or continuous. An attribute test has an attribute, relational operator and threshold. An example is classified using these attribute tests by moving the example down from the root of the decision tree, along the path that *covers* the example, *i.e.* all the attribute tests along the path of the decision tree are true for that example. When an example reaches a leaf, it is asserted to belong to the class labelled by that leaf. Figure 2.1 illustrates an example of a decision tree.

A *production rule* consists of an *antecedent* and a *consequent*. The antecedent is formed by a conjunction of conditions. For zero-order classifier learning, which represents a number of axis-orthogonal splits in the attribute space, these conditions take the form of $(A \leq v)$, $(A > v)$, $(A = v)$ or $(A \in \{v_1, \cdots, v_n\})$, where A represents an attribute and $v, v_1, \cdots, v_n$ are possible values of A. The consequent, in the case of classification systems, names a class.

An alternative representation of an organised body of information is a *production system* [77]. A production system represents an ordered list of production rules. An example is classified using these rules by moving the example from the top to the bottom of the production system, comparing the attributes of the example against the antecedents of each rule. If an example is covered by the antecedent of a production rule it is asserted to belong to the class

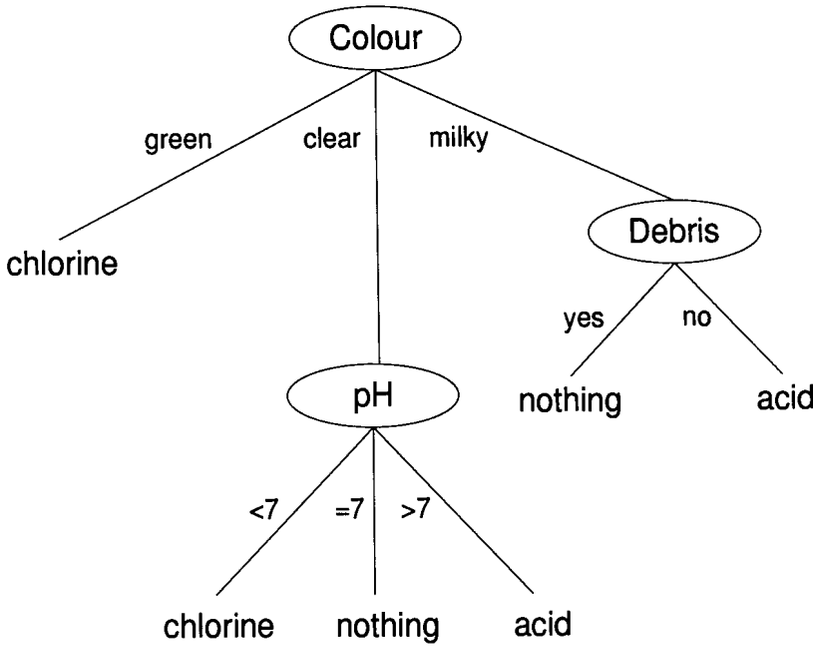

Figure 2.1: Decision tree: The lazy student's guide to swimming pool additives

labelled by the consequent. Table 2.1 shows the production system for the example of figure 2.1.

Table 2.1: Production system: The lazy student's guide to swimming pool additives

IF	$(Colour = \{green\})$	THEN	*chlorine*
IF	$(Colour = \{clear\}) \wedge (pH < 7)$	THEN	*chlorine*
IF	$(Colour = \{clear\}) \wedge (pH = 7)$	THEN	*nothing*
IF	$(Colour = \{clear\}) \wedge (pH > 7)$	THEN	*acid*
IF	$(Colour = \{milky\}) \wedge (Debris = yes)$	THEN	*nothing*
IF	$(Colour = \{milky\}) \wedge (Debris = no)$	THEN	*acid*

Two induction strategies are used to generate rules for classification systems: *selective induction* and *constructive induction*. These two approaches are described next.

Selective induction induces rules from a training set by selecting attributes from the supplied training examples, upon which the training set is split. Essentially, these attributes partition the

search space into regions that have the same class membership. In the case of decision trees, the induction algorithm utilises axis-orthogonal splits, each of which is based on a single attribute. If the supplied attributes are appropriate for representing target theories, selective induction algorithms perform well in terms of prediction accuracy and theory complexity. Some popular examples of selective induction algorithms include *ID3* [81], *C4.5* [83] and *CN2* [21][20], and are summarised below:

The ID3 algorithm uses decision trees to generate rules. ID3 generates these decision trees using a *divide-and-conquer* approach. The algorithm recursively splits a training set, P, if the training set labels consist of heterogeneous classes. The heuristic utilised to decide on a test, upon which to split the training set, is called the *gain criterion*, which is based on information theory. The gain criterion is defined as follows:

$$\begin{aligned} info(P) &= -\sum_{\psi=1}^{c} \frac{freq(C_\psi,Q)}{|Q|} \cdot log_2\left(\frac{freq(C_\psi,Q)}{|Q|}\right) \\ info_X(P) &= \sum_{\phi=1}^{o} \frac{|P_\phi|}{|P|} \cdot info(P_\phi) \\ gain(X) &= info(P) - info_X(P) \end{aligned} \tag{2.1}$$

where o is the number of outcomes of test X, Q is a set of cases belonging to some class C_ψ and c is the number of classes in the domain. The information content of the domain before splitting occurs is calculated by $info(P)$. The sum of the weighted information content of each outcome after splitting occurs is calculated by $info_X(P)$. The objective of the ID3 algorithm is to maximise the gain criterion $gain(X)$, *i.e.* to maximise the decrease in entropy of performing a split. ID3 assumes clean data and is therefore unable to deal with outliers or missing values. The trees generated by ID3 completely cover the training set. Thus, ID3 is particularly susceptible to over-fitting.

The C4.5 algorithm (latest version C5.0 [84]) is based on ID3, but has no assumptions about the purity of the data. The C4.5 algorithm uses decision trees and heuristics to generate simplified, comprehensible production rules. C4.5, like ID3, generates these decision trees using a *divide-and-conquer* approach. The heuristic utilised to perform the divide-and-conquer approach is similar to ID3's gain criterion, called the *gain ratio criterion*. The gain ratio criterion is defined as follows:

$$\begin{aligned} split\_info(X) &= -\sum_{\phi=1}^{o} \frac{|P_\phi|}{|P|} \cdot log_2\left(\frac{|P_\phi|}{|P|}\right) \\ gain\_ratio(X) &= gain(X)/split\_info(X) \end{aligned} \tag{2.2}$$

where $gain(X)$ is obtained using equation (2.1) and all variables are defined as above. The maximum information content of each outcome after splitting occurs is calculated by $split\_info(X)$. The objective of C4.5 is to maximise the gain ratio criterion $gain\_ratio(X)$, *i.e.* to maximise the relative decrease in entropy. The gain ratio criterion prevents the C4.5 algorithm from biasing toward classes with large number of patterns.

ID3 and C4.5 inherently applies to discrete data. However, in the case of continuous-valued attributes the training examples are sorted (according to the desired attribute) and a threshold is chosen that lies between any two adjacent training examples (usually the midpoint between the two examples). This threshold then acts to split the data into two subsets, according to the test $A_i < \frac{a_{i,1}+a_{i,2}}{2}$. If an attribute has n continuous values in the domain, $n-1$ splits are tested using equation (2.2) and the split with the largest gain ratio criterion is selected.

C4.5 provides a windowing strategy to reduce the number of training examples presented to the algorithm. The windowing strategy initially selects a random subset of training examples (the window) from which an initial tree is built. This tree is used to classify training examples not included in the window. The training examples that are misclassified by the initial tree are then added to the window, from which another tree is built. Quinlan's initial reason for utilising a windowing strategy was to attempt to reduce the time required to construct trees. After experimentation, however, he discovered that the windowing strategy also resulted in more accurate decision trees [83].

CN2 uses a beam search algorithm in order to find rules, and a control algorithm for repeatedly executing the search. The rules induced by CN2 are represented in the form of a production system. CN2 uses an entropy measure to decide on a set of conditions from which to induce a rule. Instead of using classification accuracy or information content as a measure of rule quality, CN2 uses the *Laplace Error Estimate* to test the significance of rules. The Laplace Error Estimate is defined as follows:

$$Laplace\_Accuracy = \frac{|Q|+1}{|R|+c} \tag{2.3}$$

where c is the number of classes in the domain, Q is a set of examples belonging to the class covered by the rule and R is the set of examples covered by the rule. CN2 also uses significance testing in order to prune the induced rules.

Constructive induction algorithms consist of two steps. One step constructs new attributes, the other generates theories. Attribute construction can be visualised as the application of

constructive operators, such as $\wedge$, $\vee$ and $\neg$, to the set of existing attributes, in order to reduce the attribute space of the problem domain. The constructive operators thus provide a mechanism for mapping an N-dimensional attribute space into an X dimensional attribute space such that $X \leq N$. Theories are generated from the X dimensional attribute space by using any of the previously mentioned selective induction algorithms. In essence, constructive induction attempts to provide a mechanism for generating complex orthogonal splits in the decision space. Zheng's *X-of-N* algorithm is an example of a constructive induction algorithm [105].

Regression systems

A *regression tree* is a decision tree that has numeric values at its terminal nodes. A *model tree*, on the other hand, is a decision tree that has multi-variate linear models at its terminal nodes. Both of these classifiers attempt to predict a numeric value associated with a class, rather than the class to which an example belongs. The objective of both model and regression trees is to perform a piecewise approximation of the instance space.

As with decision trees, an example moves along a path toward a leaf and is compared with each attribute test along the way. Once the example reaches a leaf, the example is asserted to have the target output defined by the numeric value or the multi-variate linear model. Figure 2.2 illustrates an example of a model tree.

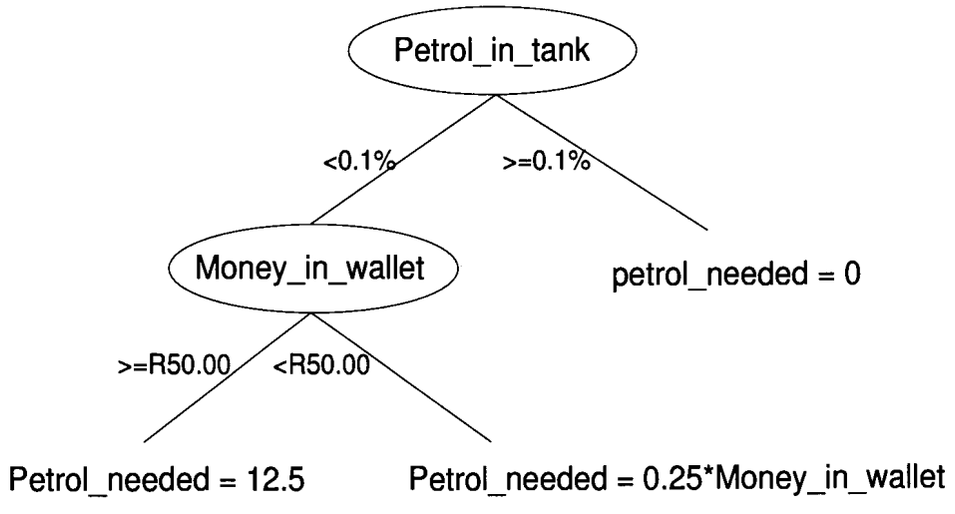

Figure 2.2: Model tree: The lazy student's guide to filling up the car

Compared with the large number of classification systems, very few regression systems are currently in existence. One of the few methods in existence is Quinlan's M5 algorithm [82]. M5 utilises model trees and is a descendant of C4.5. Like C4.5, M5 uses a heuristic in order to decide on a test. The heuristic is defined as follows:

$$\triangle error = std(P) - \sum_{\phi=1}^{o} \frac{|P_\phi|}{|P|} \cdot std(P_\phi) \tag{2.4}$$

where P, once again, represents the set of training cases, o is the number of outcomes for a test, $std(P_\phi)$ represents the standard deviation of the target values of cases in P_ϕ, and P_ϕ represents the set of patterns belonging to outcome ϕ. After examining all tests, M5 chooses the test that maximises the expected error reduction. The multi-variate linear regressions at each leaf are constructed using standard regression techniques (such as the *least squares method*) [16]. The M5 algorithm also includes heuristics for smoothing and model tree pruning. Pruning is used to remove regressions that cover outliers. Smoothing is used to adjust the values of an appropriate leaf model to reflect the predicted values at nodes along the path from the root to that leaf. Smoothing can be visualised as a process of adjusting the output of models that have few training cases or that lie on either side of an orthogonal split, where the models have very different values.

2.1.3 Past usage

Many successful applications of knowledge discovery techniques exist in the literature. This section presents a small, interesting subset of these examples which are not constrained to include only the algorithms discussed in section 2.1.2.

Spertus used C4.5 in a system called "Smokey" to dispose of abusive emails (flame mail) [95]. Smokey built a 47-element feature vector based on the syntax and semantics of each sentence, combining the vectors for the sentences in each message. The system was able to correctly classify 64% of abusive emails and 98% of normal emails.

Daelemans *et al.* applied C4.5 to the problem of natural language processing [26]. Specifically, C4.5 was applied to the formation of diminutive forms in Dutch. C4.5 proved a useful tool in corroborating or falsifying existing linguistic theories.

Adomavicius and Tuzhilin used a system called "1:1Pro" to build customer profiles [2]. Specifically, the system mined the non-alcoholic beverage sales of a number of households. As

anticipated, the system discovered that most rules pertained only to a small number of households, meaning that the system captured the idiosyncratic behaviour of individual households. The system also discovered interesting seasonal behavioural characteristics of consumers.

2.2 EVOLUTIONARY COMPUTING

This section presents a brief introduction to evolutionary computing. Section 2.2.1 introduces the reader to evolutionary computing by drawing parallels with natural evolution. The evolutionary computing definitions used throughout this thesis are discussed in section 2.2.2. A number of popular evolutionary computing paradigms are presented in section 2.2.3. Section 2.2.4 discusses a number of performance issues relating to the use of evolutionary computing. Finally, section 2.2.5 presents some of the past uses of a number of evolutionary computing paradigms. Interested readers should consult the authoritative works of Goldberg [48] and Bäck *et al.* [7][8].

2.2.1 Introduction

Evolutionary computing simulates the Darwinian principle of natural selection. Darwin discovered that a certain species of birds (finches) native to the Gálapagos islands, differed from each other in terms of beak shape [27]. He also noted that the beak varieties were associated with diets based on different foods. He concluded that when the original South American finches reached the islands, they dispersed to different environments where they had to adapt to different conditions.Over many generations, they changed anatomically in ways that allowed them to get enough food and survive to reproduce. This illustrates one of the handful of ways that species of plant, animal and bird-life evolve over long periods of time. The main concept of natural selection is that the fittest individuals in a population survive and the weakest individuals perish.

2.2.2 Definitions

Evolutionary computing (EC) utilises a population of individuals, where each *individual* represents a candidate solution to the optimisation problem. The *chromosome* associated with each

individual encodes the *genotypes* and *phenotypes* of that individual. A genotype describes the genetic or factional constitution of an individual and thus provides a mechanism to store experiential evidence. A phenotype describes the observable characteristics of an individual produced by the interaction between the *genes* and the environment. A gene represents a characteristic of the individual. The value of a gene is referred to as an *allele*.

In EC, the design of a chromosome (or individual) is paramount. The efficiency and complexity of an EC search algorithm greatly depends on the chromosome representation scheme. Traditional EC methods encode the chromosome as a binary string. More modern methods encode the chromosome as any combination of available machine types, *e.g.* real numbers, integers, characters *etc.*

The *fitness function* is the most important component of any EC paradigm. The fitness function maps a chromosome representation into a floating-point value, which quantifies the *quality* of each individual. The quality of an individual can be defined as the distance between the individual and the optimal solution. The fitness function is used to guide the other EC operators such as *selection*, *elitism*, *crossover* and *mutation*. It is important that the fitness function accurately models the optimisation problem, *i.e.* the fitness function should adequately model the solution space to a problem. Obviously, a fitness function that inaccurately models the optimisation problem could lead to sub-optimal solutions. Also, the fitness function should reflect all the criteria to be optimised, *e.g.* certain problems require the optimisation of both the output and the internal architecture of a solution. Constraints on the problem space can also be incorporated into the fitness function through penalisation of those solutions that violate the constraints. However, constraints can also be directly applied to the EC operators.

Before beginning the evolutionary process, an initial population of individuals must be generated. A standard way of generating these individuals is to randomly set each gene in a chromosome. The goal of random selection is to uniformly represent the entire search space. If prior knowledge is available on the search space, heuristics can be used to bias the initial population toward potentially good solutions. This, however, may lead to premature convergence because the entire search space is not covered.

As was mentioned earlier, there are a number of other operators involved in an EC optimisation algorithm. Selection operators emphasise better solutions in a population. Many selection schemes were compared by Goldberg and Deb [49]. *Random selection* selects indi-

viduals with no reference to fitness at all. *Proportional selection* selects an individual proportionately according to its fitness value, using roulette wheel sampling. *Tournament selection* selects a group of n individuals to take part in a tournament and the best individual is selected. *Rank-based selection* uses the rank ordering of fitness values to determine the probability of selection and not the fitness itself.

Elitism involves the selection of a set of individuals from the current generation to survive to the next. The individuals that survive to the next generation can be selected using the previous selection operators or as the best individuals from the current generation.

Crossover operators model reproduction in nature. Superior individuals should have more opportunities to reproduce. An extreme example of this can be seen in nature in wolf packs, where only the alpha male and the alpha female have pups, because the alpha male and alpha female actively terminate pups produced by other individuals. Each generation is thus strongly influenced by the genes of the fitter individuals.

Crossover is achieved by combining genetic material from a number of parents (usually two) to create a new individual. A number of crossover strategies exist for binary string representations and are also applicable to other representations. *Uniform crossover* begins by creating a random binary mask. This mask is applied to both parents in order to generate a new individual,

$$\mathbf{c} = (\mathbf{m} \wedge \mathbf{a}) \vee (\neg\mathbf{m} \wedge \mathbf{b})$$

where $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and $\mathbf{m}$ respectively represent the vectors of parent A, parent B, the new individual C and the mask. *One-point crossover* selects a random bit position. All bits in the first parent before the bit position and all bits in the second parent after the bit position are placed in the target chromosome. *Two-point crossover* is similar to one-point crossover, except that two bit positions are chosen. The three crossover strategies mentioned above are shown in figure 2.3.

Mutation operators serve to expand the search space of the EC optimisation algorithm by injecting new information into the search space. Mutation is thought to occur in nature due to cosmic and other types of radiation, which damages the molecules found in DNA [87]. Mutation in EC is performed by randomly injecting new genetic material into an individual, thus damaging the genes of the chromosome. Mutation is performed on an individual C by randomly adjusting a number of the genes in the individual's chromosome. Two popular mutation strategies for binary string representations are *random mutation*, which randomly selects a

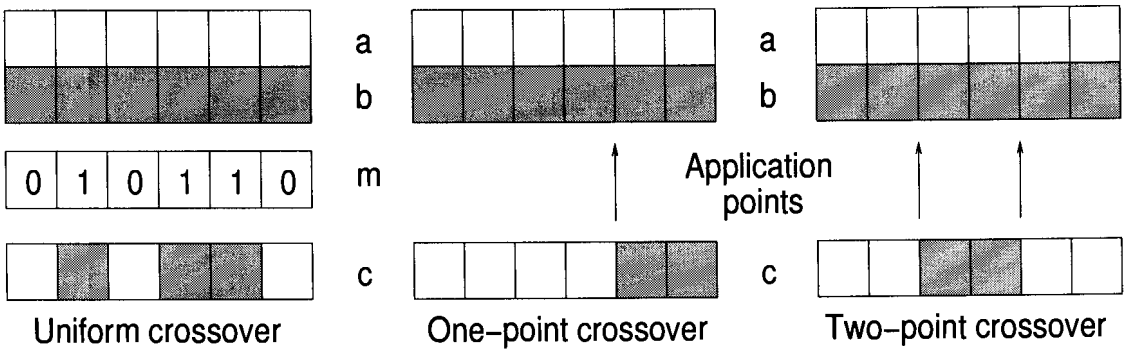

Figure 2.3: Illustration of crossover operators for binary strings

number of bits and negates them, and *inorder mutation*, which selects pairs of bit positions and randomly negates bits between pairs of bit positions. The two mutation strategies mentioned above are shown in figure 2.4

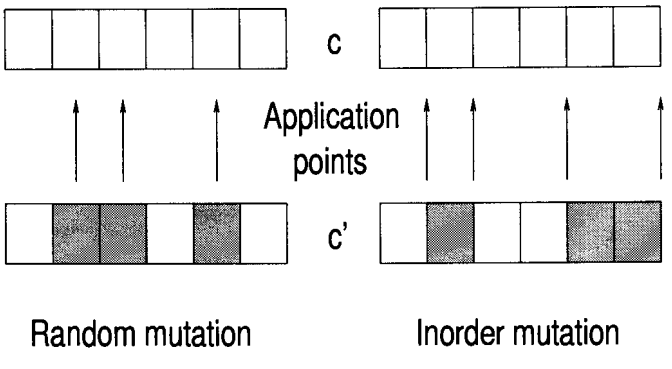

Figure 2.4: Illustration of mutation operators for binary strings

The general EC optimisation algorithm proceeds as follows:

1. Initialise the generation counter $g := 0$.
2. Initialise the initial population G_g.
3. Evaluate the fitness of each individual $G_{g,n} \in G_g$.
4. Set $g := g + 1$.

5. Select elite individuals $E_g \subset G_{g-1}$.
6. Select parents from the previous generation G_g.
7. Perform crossover to form offspring O_g.
8. Mutate individuals in O_g.
9. Set $G_g = E_g \cup O_g$.
10. If the stopping criteria has not been fulfilled, return to 3.

where the stopping criteria can be any of the following:

- Stop when the maximum number of generations is reached.
- Stop when all the individuals in the population are homogeneous.
- Stop when a desired level of fitness is achieved.

2.2.3 Paradigms

There are many evolutionary computing paradigms. This section overviews some of the most important ones.

Genetic algorithms (GAs) model genetic evolution. The original GAs, introduced by Holland, used a bit string representation, proportional selection and crossover as the primary method to produce more individuals [55]. A number of changes have been made to the original GA including representation, selection, elitism, mutation and crossover changes.

Island genetic algorithms model the migration of individuals to other subpopulations/islands [18]. Essentially, each island represents a subpopulation of the entire population of individuals. Individuals compete for survival on each of these subpopulations, but are also allowed to migrate to other subpopulations. The subpopulations can also represent subsets of the search space or different optimisation criteria.

Genetic programming (GP), introduced by Koza [68], is a specialisation of GAs that utilises a tree-based chromosome representation scheme. This tree-based representation can be used to represent programs or expressions. Each individual in a GP population, therefore,

represents a possible program or expression, which is an element of the program space formed from all possible programs that can be created from a given grammar.

Evolutionary programming, introduced by Fogel [39], differs from the GA and GP paradigms, in that evolutionary programming models phenotypic evolution. The evolutionary process attempts to find a set of optimal behaviours from the space of observable behaviours.

Evolutionary strategies, introduced by Rechenberg [85], models the evolution of evolution. The evolutionary process attempts to evolve both the genetic characteristics of an individual as well as a set of strategy parameters. The strategy parameters control the evolutionary characteristics of the individuals.

Coevolution models the complementary evolution of closely related species [7]. Two coevolutionary processes are *symbiotic* relationships, where two species co-operate for their mutual benefit, and *predator-prey* relationships, where each species attempts to out-perform the other in their environment. Coevolution does not define optimality through the use of a fitness function, but defines optimality as the overall success of one population over another, *i.e.* a relative fitness. Coevolution has mostly been applied to two-agent games, where the objective is to evolve a game strategy.

Cultural evolution, introduced by Reynolds [86], models societal evolution. Two search spaces are maintained by cultural evolution: the *population space* and the *belief space*. The belief space models the cultural behaviours of a population. An *acceptance function* is used to determine which individuals have an influence on the current beliefs. The belief space is then adjusted with the experiential knowledge of influential individuals. The belief space is then used to influence the individuals in population space.

2.2.4 Performance issues

The representation of a chromosome or solution in an evolutionary computing (EC) algorithm has implications for the performance of that algorithm. Binary coding, although frequently used, introduces *Hamming cliffs* into the search space. A Hamming cliff is formed when two numerically adjacent values have bit representations that lie far apart. This represents a problem when a small change in variables should result in a small change in fitness. An alternative representation is Gray coding, where the Hamming distance between successive numerical values is 1.

Tree-based representation schemes, as found in genetic programs, can lead to discontinuous or inefficient results, *e.g.* if a grammar contains the natural logarithm function, values in the domain $(-\infty, 0]$ will have no meaning. Alternatively, if a grammar consists of boolean connectives, expressions such as $\neg\neg\neg\neg n$ will not be unlikely. Additional semantic tests need to be included to ensure the semantic correctness of each individual tree.

Mutation operators inject new genetic material into a population. However, Hinterding *et al.* argued that mutation operators are more than just background operators and are crucial to the success of any EC algorithm [53]. Mutation operators should, in fact, utilise *domain specific knowledge*. Domain specific knowledge is any previously discovered knowledge that pertains to the specific optimisation problem of the EC, *e.g.* if the optimisation problem is the design of wings for a fixed wing aircraft, domain specific knowledge would include the known strengths of various compounds from which a wing can be manufactured.

The argument for the use of domain specific knowledge is supported by the *No Free Lunch theorem* of Wolpert and Macready [103], which states that

> "$\cdots$ the average performance of any pair of algorithms across all possible problems is exactly identical."

and further states that

> "$\cdots$ if some algorithm a_1's performance is superior to that of another algorithm a_2 over some set of optimisation problems, then the reverse must be true over the set of all other optimisation problems."

From the above example, if an algorithm has specifically been engineered for the design of wings for a fixed wing aircraft, the algorithm may not necessarily perform well at optimising the architecture of an artificial neural network. However, because the algorithm was only engineered to solve the wing design problem, it is not expected to also optimise artificial neural network architectures.

The mutation rate of an EC algorithm directly controls the injection rate of new material into the population and therefore controls how rapidly the search space is broadened. A large mutation rate causes the algorithm to behave more like a random search, because experiential knowledge gained from reproduction is lost. When the experiential knowledge is lost, the algorithm continually searches through parts of the search space it has already visited, leading

to increased training time. A small mutation rate can cause the algorithm to stagnate in a local minimum, because all of the individuals in a population become homogeneous. Stagnation thus results in poor training accuracy and sub-optimal convergence.

The choice of the initial population size has implications for the performance of an EC algorithm. Large populations may be more computationally complex than small populations, but they cover more of the search space.

Elitism can ensure that an EC algorithm retains good genetic material. However, the elite group can lead to stagnation, because the elite individuals are more likely to be involved in crossover. If an elite group survives for many generations, the population will become homogeneous, leading to sub-optimal convergence.

The crossover rate, on the other hand, controls how rapidly the search space is reduced. A large crossover rate leads to a large number of individuals being involved in reproduction. This can cause slower convergence, because increases in fitness in one individual take a long time to filter through to other individuals in a population. A small crossover rate, on the other hand, leads to a small group of individuals being involved in reproduction. This can cause the algorithm to stagnate in a local minimum, because, as with a low mutation rate, all the individuals in a population rapidly become homogeneous.

Additionally, the population size, generation gap (elitism), mutation rate, and crossover rate all exhibit some or other level of dependence on one another and on the problem domain. The correct parameter choices are thus fairly problem specific.

2.2.5 Past usage

Evolutionary computing paradigms are increasingly being applied to situations that are computationally complex or situations where humans have very little prior knowledge of the problem domain. Examples include routing, network optimisation, design, game strategies *etc.* This section overviews just some of the applications.

Biles used a genetic algorithm for generating improvised jazz solos [12]. The method primarily received an input chord progression, as well as other cues, from which the algorithm attempted to improvise a melody. The genetic algorithm used a multi-objective fitness function in order to reward solutions that were considered musically correct. Although Biles' attempts were successful, he did concede that the method was far from musically perfect.

Beretta *et al.* used a genetic algorithm to perform fuzzy compression and decompression of an input image [11]. Specifically, the method optimised the fuzzy membership boundaries that described a patch of pixels. The method was found to perform well on a large database of images and had very good generalisation ability, specifically with regards to preserving important features within a picture.

Göckel *et al.* used a hybrid genetic algorithm to solve the channel routing problem [47]. The algorithm used domain specific knowledge during initialisation of the population to improve the efficiency of the algorithm. The algorithm provided improved results over similar attempts and could feasibly be implemented on very large channels.

Koza used a coevolutionary genetic program to evolve game strategies for a number of types of strategy games [67]. Specifically, the method attempted to evolve a strategy equivalent to the minimax strategy, without prior knowledge of the minimax strategy. The algorithm was successful in discovering the minimax strategy.

Schweitzer *et al.* used evolutionary strategies to optimise the layout of a road network [89]. The method minimised the cost of road construction and maintenance, while providing direct connections between nodes in order to avoid detours. Evolutionary strategies proved to be a capable tool in solving this frustrating problem.

2.3 ARTIFICIAL NEURAL NETWORKS

This section presents a brief introduction to the machine learning paradigm of artificial neural networks. Section 2.3.1 introduces the reader to artificial neural networks by drawing parallels with biological neural systems. The fundamentals of artificial neural networks are presented in section 2.3.2. Performance issues pertaining to the use of artificial neural networks are discussed in section 2.3.3. Finally, section 2.3.4 presents the past usage of artificial neural networks. The interested reader is encouraged to review the works of Zurada [106] and Bishop [13].

2.3.1 Introduction

Artificial neural networks are an attempt to model biological neural systems. One of the primary features of biological neural systems is that they can learn from their environments.

For example, an animal can learn to identify possible sources of food and water, and also to identify potential threats. Also, many types of birds in infancy have the ability to learn to recognise the individual calls of their parents, particularly through the vast cacophony created by other members of their species in breeding areas.

The primary building blocks of biological neural systems are called neurons. A neuron consists of a cell body which contains a nucleus and cytoplasm, from which threadlike processes called dendrites extend. Electrochemical impulses travel from the cell body along a single fibre, called an axon, to other neurons. The processes of one neuron never touch those of another neuron and are separated by a space called a synapse. A synapse serves to either inhibit or propagate an electrochemical signal.

2.3.2 Definitions

An artificial neuron (AN) is a model of a biological neuron. An AN represents a non-linear mapping of I real-valued inputs ($\Re^I$) to a real-valued output ($[0,1]$,$[-1,1]$ or $[-\infty,\infty]$). Each input x_i is associated with a weight w_i, which serves to amplify the aforementioned input. Each AN has an associated *activation function* f_{AN} and threshold value θ, both of which control the output of the AN. The output of an AN is computed as

$$f_{AN}(net) = \sum_{i=1}^{I} (x_i w_i) - \theta$$

for a *summation unit* and computed as

$$f_{AN}(net) = \prod_{i=1}^{I} (x_i w_i) - \theta$$

for a *product unit*.

Different types of activation functions f_{AN} can be used. In general, activation functions are monotonically increasing mappings, where $f_{AN}(-\infty) = 0$ or $f_{AN}(-\infty) = -1$, and $f_{AN}(\infty) = 1$. Frequently used activation functions include the linear, step, ramp, sigmoid, hyperbolic tangent and Gaussian functions, of which the sigmoid activation function is the most common. The sigmoid activation function is defined as follows:

$$f_{AN}(net) = \frac{1}{1+e^{-\lambda net}}$$

where λ controls the slope of the function. Usually $\lambda = 1$, which gives an active domain (where $f'_{AN} \not\approx 0$) for the sigmoid function of $[-\sqrt{3}, \sqrt{3}]$.

The weights and threshold values of an AN are obtained through learning. A learning rule, such as the popular *gradient descent* learning rule, is used to train an AN [99]. Other learning rules include *Widrow-Hoff* [17], *generalised delta* [74], *etc*. The gradient descent learning rule requires the definition of an error metric to measure the AN's error in approximating a target output. The *sum squared error*

$$E_{SS} = \sum_{l=1}^{|P|} (y_{\rho_l} - y^{\bullet}_{\rho_l})^2$$

is usually used, where y_{ρ_l} and $y^{\bullet}_{\rho_l}$ represent the target and predicted outputs for pattern ρ_l, and $|P|$ is the size of the training set.

An AN essentially represents an I-dimensional hyperplane, where the curvature of the hyperplane is described by the AN activation function. This hyperplane is of importance, because it can be used either to describe a decision boundary between two linearly separable classes or to fit a regression curve through a number of data-points.

An artificial neural network (ANN) is a layered interconnection of ANs. A wide variety of ANN architectures are in use today [13][106]. A *feed-forward ANN*, illustrated in figure 2.5, consists of an input layer, a number of hidden layers and an output layer, where each layer is connected to the next in the aforementioned order. A feed-forward ANN can also implement direct connections between the input and output layer. Feed-forward ANNs are important, because it has been proved that feed-forward ANNs with monotonically increasing differentiable functions can approximate any continuous function with one hidden layer, provided that the hidden layer has enough hidden neurons [57][58]. *Functional link ANNs* are feed-forward ANNs that implement activation functions in all layers including the input layer [79]. *Recurrent ANNs* have feedback connections, which allow the ANN to learn the temporal characteristics of a given dataset [15]. The *Elman recurrent ANN* copies hidden units into a context layer, which is then fed back to the hidden layer. The *Jordan recurrent ANN* copies output units into a state layer, which is then fed back to the hidden layer.

Generally, ANNs (like other empirical learning algorithms) can be categorised into two types according to their learning strategies: supervised and unsupervised. Supervised ANNs attempt to predict the target response of a given input pattern. Input patterns can either

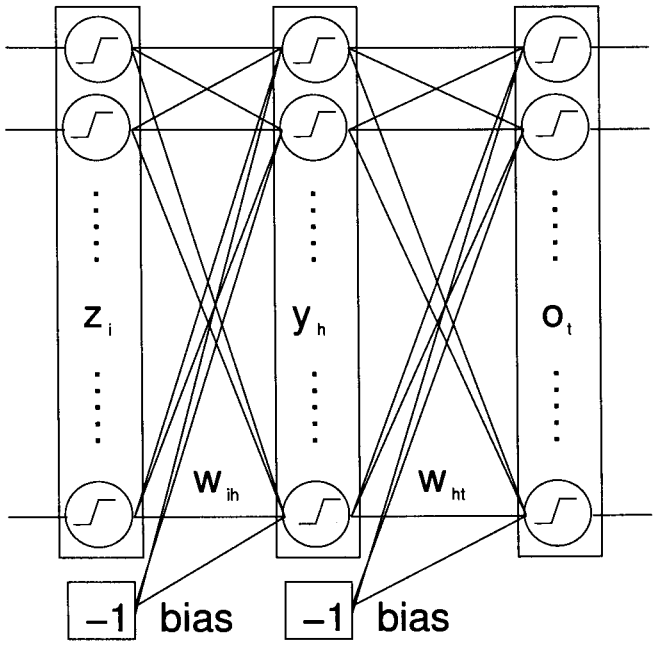

Figure 2.5: Feed-forward artificial neural network illustration

form part of a classification or regression problem. Unsupervised ANNs attempt to discover patterns, features or relationships between input patterns, without supervision, and are mainly used to perform clustering. Unsupervised ANNs will be discussed further in section 2.4.

Supervised ANNs utilise a training set consisting of a number of input vectors, where each input vector has an associated target vector. The ANN uses the target vector to determine the overall error of the network. This overall error is employed by a learning rule to either directly adjust the ANN's weights and thresholds or as an error estimate for some optimisation method. ANN optimisation methods can either be classed as local or global. Examples of local optimisation methods include the popular *gradient descent with back-propagation* [99] and *scaled conjugate gradient* [75]. Examples of global optimisation methods include *leapfrog* optimisation [60], *genetic algorithms* [3] and *particle swarm optimisation* [61]. Local optimisation methods can, in turn, be categorised according their weight update strategies:

- **Stochastic** learning adjusts the weight and threshold values of an ANN after each training pattern presentation.

- **Batch** learning adjusts the weight and threshold values of an ANN only after the presentation of all training patterns.

The gradient descent optimisation algorithm can be divided into two phases. The feed-forward phase calculates the output of the ANN. The back-propagation phase propagates an error signal back from the output layer, through the hidden layer, to the input layer. Essentially, the back-propagation phase adjusts the weights of each AN in the ANN proportionately according to the amount of error introduced by that AN. The weight update equations of an ANN have to be derived separately for each type of activation function employed by the ANN and each optimisation algorithm. An important aspect to the gradient descent algorithm is the introduction of a momentum term α and learning rate η to the weight update equations. These terms control the convergence speed of an ANN and will be discussed in section 2.3.3.

Each training iteration (referred to as an epoch) of an ANN represents one pass through the training set. After each epoch the *mean squared error* is calculated as

$$E_{MS} = \frac{\sum_{l=1}^{|P|} \sum_{t=1}^{T} (y_{t,\rho_l} - y^{\bullet}_{t,\rho_l})^2}{|P|T}$$

where T represents the number of target outputs for the ANN.

Stochastic learning can be summarised with the following pseudo-code algorithm, assuming gradient descent:

1. Initialise the weights for each layer to random values, *e.g.*

$$w_{ih} := U\left(\frac{-1}{\sqrt{I+1}}, \frac{1}{\sqrt{I+1}}\right), i \in \{1, \cdots, I+1\}, h \in \{1, \cdots, H\}$$

$$w_{ht} := U\left(\frac{-1}{\sqrt{H+1}}, \frac{1}{\sqrt{H+1}}\right), h \in \{1, \cdots, H+1\}, t \in \{1, \cdots, T\}$$

   where I, H and T are the number of input, hidden and output units respectively (the number of weights for each AN is increased by one to cater for the bias). Also initialise the learning rate η, the momentum α and the number of epochs ε.

2. Initialise the mean squared error $E_{SS} = 0$.

3. For each training pattern $\rho_l \in P$

(a) Perform the feed-forward phase to calculate y_{t,ρ_l} for each output AN, t.

(b) Calculate the error signal $\sigma_{o_{t,\rho_l}} = (y_{t,\rho_l} - y^{\bullet}_{t,\rho_l})^2$ for each output AN.

(c) Perform back-propagation of the error signal $\sigma_{o_{t,\rho_l}}$, by adjusting the output layer weights w_{ht}, computing the hidden layer error signals $\sigma_{y_{h,\rho_l}}$ and adjusting the hidden layer weights w_{ih}.

(d) Set $E_{SS} := E_{SS} + \sum_{t=1}^{T}(y_{t,\rho_l} - y^{\bullet}_{t,\rho_l})^2$

4. Set $\varepsilon := \varepsilon + 1$

5. Set $E_{MS} := \frac{E_{SS}}{|P|T}$

6. If the any of the stopping criteria are unsatisfied, goto step 2.

The stopping criteria usually includes:

- Stop when the maximum number of training epochs is reached.
- Stop when the mean squared error is small enough.
- Stop when over-fitting is observed.

2.3.3 Performance issues

As with all algorithms, a number of factors influence the performance of an artificial neural network (ANN). The performance of an ANN is influenced by three competing objectives, *i.e.* accuracy, complexity and convergence. Accuracy is concerned with those aspects of an ANN that impede its generalisation ability. Complexity is concerned with such issues as the ANN architecture, the size of the training set and the complexity of the optimisation method. Convergence is concerned with the stability of an ANN, *i.e.* whether the variability in the ANN outcome is within acceptable levels. This section discusses ANN *data preparation*, ANN *weight initialisation*, the *learning rate, momentum and optimisation method* of an ANN, ANN *architecture selection* and *active learning* with respect to how they relate to the above mentioned objectives.

Data preparation

ANNs usually require the scaling of their input data. Incorrect scaling leads to decreased ANN accuracy, when the scaled inputs do not cover the entire active domain of the ANN activation functions. Patterns should thus be scaled to the active domains of the activation functions utilised in the hidden and output layers. Also, depending on the activation functions utilised in the output layer, the outputs of an ANN will have to be scaled. It is also important to ensure that the data is numeric, *i.e.* nominal attributes need to be converted into a continuous representation. A nominal attribute with n different values is recoded as n binary valued inputs, where the input parameter corresponding to a particular nominal value is assigned the value of 1 and the remainder are assigned the value of 0.

Outliers can significantly affect the weights of an ANN and can lead to decreased ANN accuracy in terms of generalisation. Essentially, an outlier has a large effect on the sum squared error metric, utilised by many ANN optimisation algorithms. Such optimisation algorithms will adjust weights so as to minimise the sum squared error. Thus, a single pattern can exert a disproportionate influence on the weights of an ANN. This leads to a case where the training error of an ANN might be good, but the generalisation error is poor. Thus, outliers result in a bias of the weights of the ANN toward the outlier. A number of strategies exist for handling the outlier problem. One solution, is to remove the outliers before training, using statistical techniques. Another is to use a robust objective function, unlike the sum squared error, that is not influenced by outliers.

Training patterns with missing attribute values may contain useful information. Removal of these training patterns could result in decreased ANN accuracy, particularly if the pattern occurs in a region of low data point density. A common solution is to replace the missing value with the average value of the attribute.

ANN complexity ultimately depends on the size of the training set utilised during ANN training. A number of strategies have emerged to control large and, conversely, small training sets. The introduction of noise, sampled from a normal distribution, in small training sets has been shown to result in reduced training time and increased accuracy [56]. Several researchers have also developed strategies for coping with large training sets that involve the presentation of various subsets of the data to the ANN training process. This section differs from the later *active learning* section in that the ANN has no active control over the subsets presented to

it. These training set manipulation strategies include *selective presentation* [78], *incremental training strategies* [24], *increased complexity training* [22] and *delta subset training* [23].

The selective presentation strategy divides the original training set up into two other training sets. One set containing "typical" patterns, and the other containing "confusing" patterns. Typical patterns refer to patterns that lie far away from decision boundaries and, conversely, confusing patterns refer to patterns that lie close to decision boundaries. The two new training sets are alternately presented to the training algorithm. In practise, this algorithm is not practical because prior knowledge of the search space is required in order to divide the data into subsets.

The incremental training strategies start with a small random initial subset. During training additional patterns from the original set are added to the actual training set. This algorithm is practical because it assumes no prior knowledge of the search space.

The increased complexity training strategy splits the original training set up into subsets of increased difficulty. The strategy starts by presenting easy problems to the learning algorithm, and gradually increasing the level of difficulty. A drawback of the method is that the complexity measure of the patterns is problem dependent.

The delta subset training strategy orders the training patterns according to their inter-pattern distance. The metric used to perform the ordering can either be the Hamming distance or the Euclidean distance. The learning algorithm is then presented with either the smallest difference patterns first or the the largest difference patterns first.

Weight initialisation

Gradient-based optimisation methods are very sensitive to the initial weight vectors. This sensitivity can lead to poor convergence. A particularly poor strategy is the initialisation of these vectors to 0, which can be shown to be equivalent to an ANN with only one hidden unit. A good strategy is to set the initial weight vectors to random weights in the range $[\frac{-1}{\sqrt{connections}}, \frac{1}{\sqrt{connections}}]$, where the *connections* parameter represents the number of connections leading to an artificial neuron (AN) [100].

Learning rate, momentum and optimisation method

The learning rate η controls the step size of the ANN optimisation method. A small learning rate results in small weight adjustments and a large learning rate, conversely, results in large weight adjustments. The learning rate parameter introduces a trade-off between the speed of convergence and the accuracy of convergence. This trade-off exists, because small weight adjustments cause the ANN to take longer to stabilise. Large weight adjustments, on the other hand, may cause the ANN to wildly oscillate between weight values and jump over a local minimum.

The momentum term α prevents unlearning in stochastic learning. Unlearning occurs when two successive weight updates result in no change in the state of the ANN weights, *i.e.* when the second weight update negated the effect of the first weight update. Momentum in ANNs is similar to inertia in physics, in that updates to the weights have little effect, unless they are sustained.

The optimisation method employed by an ANN has a significant influence on performance. While gradient descent is very popular, it suffers from slow convergence and susceptibility to local minima. However, optimisation methods such as *particle swarm optimisation* and *genetic algorithms* are significantly more computationally expensive. Thus, a trade-off exists among various types of optimisation methods.

Architecture selection

Learning in ANNs does not just include finding the optimal weight values, it also includes finding the optimal ANN architecture. Finding the optimal architecture is crucial, because a large number of weights, trained for too long, with noise in the training data causes an ANN to "memorise" that data. Memorisation results in poor ANN generalisation ability. Finding the optimal architecture ultimately requires a search over all possible architectures. The optimal architecture of an ANN is thus that architecture which results in the best generalisation performance. Architecture selection can be divided into three categories:*regularisation*, *network construction* and *network pruning*.

Regularisation involves the addition of penalty terms to the ANN objective function, which penalises network complexity (network size). A large number of regularisation strategies exist in the literature [46][63][98][101].

Network construction involves the growth of a small network by dynamically adding hidden ANs during training. This method requires the ANN to analyse and decide on an appropriate time to add new hidden ANs. Deciding on the appropriate time is, however, not trivial and can result in over-fitting and increased training time [42][54].

Starting with a too large architecture, network pruning involves the removal of unnecessary ANs either during or after training. The decision to prune an AN depends on some measure of the relevance of that AN. A large number of pruning techniques exist in the literature. *Optimal brain damage* is a popular technique, that uses sensitivity analysis to remove redundant weights [70]. Variants of optimal brain damage include *optimal brain surgeon* [52] and *optimal cell damage* [19]. Fletcher *et al.* utilise the Fisher information matrix as well as statistical hypothesis testing to determine the optimal number of hidden units and weights [38]. Engelbrecht used sensitivity analysis in order to prune irrelevant weights, hidden units and input units [31].

Active learning

Most ANNs are passive learners, *i.e.* they have no control over the training data presented to them. Active learning, on the other hand, allows ANNs to make optimal use of the training data. The ANN trains on the patterns it regards as most informative, by automatically removing patterns that are redundant or ambiguous from the training set. This section differs from training set manipulation in that the ANN has active control over the data. There are two main approaches to active learning: *incremental learning* and *selective learning*.

Incremental learning starts with an initial subset of the training data. At specified intervals during training, further patterns are selected from the training set using some or other heuristic. As training progresses the size of the actual training set increases [25][35][43][71]. Selective learning differs from incremental learning in that training starts with the full training set and patterns are discarded as they are found to be redundant [33][34]. Engelbrecht and Brits present an interesting active learning strategy that makes use of clustering to select the most informative patterns for training [32].

2.3.4 Past usage

Artificial neural networks (ANNs) are increasingly being applied to diverse fields such as feature recognition, compression, design, forecasting, classification *etc.* This section presents a small subset of these applications.

Le Cun *et al.* used an ANN for the recognition of hand-written digits on a database of zip-code examples provided by the U.S. Postal Service [69]. The algorithm was fairly accurate (1% error rate) and required minimal preprocessing of the input data. The results indicated that the method was extensible to alphanumeric characters. Indications were that the method was also comparitively fast, with the ability to recognise over 10 digits per second.

Fanghanel *et al.* used an ANN in the field of data compression to find the optimal parameters for Wavelet Transform Coding [37]. Specifically, the ANN learnt the optimal Daubechies filters for different one dimensional signals. Their findings indicated that the ANN resulted in better compressed outputs than the standard decompositions under noisy circumstances.

Sellar *et al.* used an ANN approach to assist in the design of a "hovercraft" [90]. The method was required to select an appropriate combination of design variables, in order to come up with a feasible design. The design variables consisted of a number of discipline-specific local optimisations. An ANN based optimisation algorithm was employed in order to optimise the global combination of design variables. Sellar *et al.* termed these combinations, response surface approximations. The method was shown to reduce the cost and time associated with designing a system, in comparison to the traditional methods of designing such a system.

Yao *et al.* used a standard back-propagation ANN in order to forecast exchange rates between the Swiss Franc and the U.S. Dollar [104]. The method modelled different trading strategies and computed the average paper profits for adopting any given strategy. Depending on the strategy employed, they showed that average paper profits between 11.36% and 27.59% could be achieved over different time horizons.

2.4 CLUSTERING

This section presents a brief overview of clustering and various clustering techniques. Section 2.4.1 introduces the goals of clustering, as well as the main categories into which all clustering algorithms fall. K-means clustering is discussed in section 2.4.2. Section 2.4.3 presents the

learning vector quantiser. Self organising maps are discussed in section 2.4.4. Finally, section 2.4.5 presents the split-and-merge algorithm.

2.4.1 Introduction

Clustering is the process of finding groups of similar data points in a given dataset, *i.e.* finding regions in a dataset with a high data point density. Essentially, any clustering algorithm attempts to reduce the variance among data points in each of its constituent clusters. Clustering can thus be seen as a process of partitioning a set of data points, such that each of the subsets of those data points is homogeneous with respect to some characteristic. Each cluster represents knowledge about its constituent data points.

Clustering methods can broadly be classified into two main categories [5][51][66]:

- **Partitional clustering** aims to directly partition a given dataset into disjoint subsets (clusters) such that specific clustering criteria are optimised.
- **Hierarchical clustering** proceeds successively by merging smaller clusters into larger ones.

Clustering is primarily used to reduce the amount of data to be presented to machine learning algorithms. Clustering is also important for *exploratory data analysis* [28][64], where little is known about a dataset or problem. Exploratory data analysis attempts to make a set of data points more accessible by using simple analysis methods to provide a preliminary overview of the content of those data points.

2.4.2 K-means clustering

K-means clustering is a fast, rough, partitional clustering algorithm [72]. K-means clustering treats each training pattern in a dataset as a real-valued input vector $\rho_l, l \in \{1, \cdots, |P|\}$, where $|P|$ is the size of the dataset. Nominal attributes need to be converted to binary-valued features as described in section 2.3.3. K-means clustering initialises k buckets (or clusters) $C_\delta, \delta \in \{1, \cdots, k\}$, where each bucket is associated with a centroid vector $\mathbf{w}_\delta$. Each centroid vector $\mathbf{w}_\delta$ represents the average of all of the input vectors in C_δ.

The algorithm starts by randomly removing k training patterns from the dataset and inserting one pattern into each bucket. The algorithm proceeds by placing each of the remaining training patterns in the dataset into the bucket whose centroid lies closest to that training pattern. The distance metric used by the algorithm can either be the *Manhattan Distance* or the *Euclidean distance*. After all training patterns have initially been clustered, the algorithm repeatedly iterates over all the training patterns in all buckets, moving each training pattern to the bucket with the closest centroid, until a stopping criterion is reached.

The k-means clustering algorithm proceeds as follows:

1. Initialise k centroids $(\mathbf{w}_1, \cdots, \mathbf{w}_k)$ such that each centroid is initialised to one input vector

$$\mathbf{w}_\delta = \rho_\delta, \delta \in \{1, \cdots, k\}$$

   where each cluster C_δ is associated with the centroid $\mathbf{w}_\delta$.

2. For each input vector $\rho_l, l \in \{1, \cdots, |P|\}$

   (a) Find the nearest centroid $\mathbf{w}_{\delta_*}$, *i.e.* if

$$\|\rho_l - \mathbf{w}_{\delta_*}\| \leq \|\rho_l - \mathbf{w}_\delta\|,$$

$$(\delta = 1, \cdots, k) \wedge (\delta \neq \delta_*)$$

   then

$$C_{\delta_*} = C_{\delta_*} \cup \{\rho_l\}$$

3. For each cluster C_δ, where $\delta \in \{1, \cdots, k\}$

   (a) Update the cluster centroid $\mathbf{w}_\delta$ to be the centroid of all the samples currently in C_δ using

$$\mathbf{w}_\delta = \frac{\sum_{\rho_l \in C_\delta} \rho_l}{|C_\delta|}$$

4. Compute the quantisation error

$$E_Q = \sum_{\delta=1}^{k} \sum_{\rho_l \in C_\delta} \|\rho_l - \mathbf{w}_\delta\|^2$$

5. Return to step 2 until E_Q does not change significantly, cluster membership does not change or a maximum number of iterations is reached.

Many k-means clustering variants exist in the literature. Alsabti *et al.* present an interesting k-means clustering algorithm that makes use of a tree structure in order to reduce the number of prototype comparisons during each training cycle [4]. Basically, the tree structure represents a nearest neighbour ordering of the training patterns. *ISODATA* is another k-means clustering variant suitable for image clustering [9].

2.4.3 Learning vector quantisers

A learning vector quantiser (LVQ) is a neural network based, unsupervised, partitional clustering algorithm [65]. An LVQ has two layers: an input layer and an output layer. The LVQ training process constructs clusters based on competition between output artificial neurons. Each output unit of the LVQ represents a cluster (not to be confused with a classification). During training, the output unit whose weight vector is closest to the current input pattern is declared the winner. The weights of the winning output unit and that of its neighbours are adjusted to better resemble the training pattern. The distance between an input pattern and a weight vector is measured using the *Euclidean distance*.

In essence, the LVQ is very similar to the k-means clustering algorithm. Each output unit's weights can be viewed as a centroid vector, and the update equations cause those centroid vectors to move in order to cover or describe a number of patterns.

2.4.4 Self-organising maps

Self-organising maps (SOMs) were motivated by the self-organisation characteristics of the human cerebral cortex, such as the visual cortex and the auditory cortex [66]. The SOM is a multi-dimensional scaling method to project an $|P|$-dimensional input space to a discrete output space of lower dimension. This discrete output space is usually a two-dimensional grid, but can be toroidal. The SOM uses the grid to approximate the probability density function of the input space, while still maintaining the topological structure of the input space. Therefore, if two input patterns are close to one another in the input space, then the patterns will also be close together in the SOM.

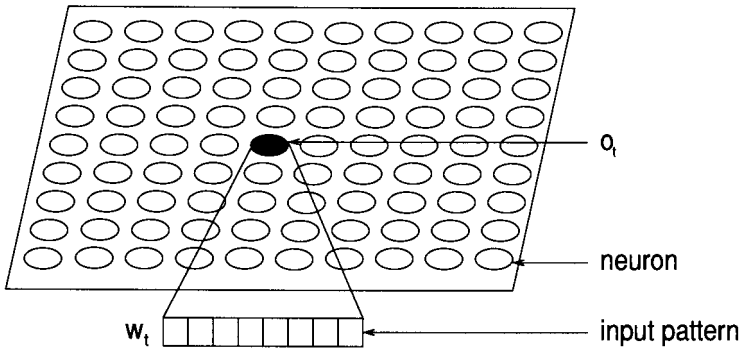

Figure 2.6: Self-organising map architecture

SOM training is based on a competitive learning strategy. Each artificial neuron (AN) o_t on the map is associated with an I-dimensional weight vector $\mathbf{w}_t$ (shown by figure 2.6). Two types of clustering occur for SOMs:

- The first is during training where input patterns are mapped to the closest AN; so, the AN serves as a centroid of a cluster of geometrically similar patterns.
- Then, after training, centroids/neuron vectors are further clustered together to form groupings of similar ANs.

Training starts by initialising each AN's weight vector. The weight vector can be initialised to random values, to a randomly selected input pattern or any other suitable strategy. Each training pattern is presented to the SOM and the AN with the shortest *Euclidean distance* to this pattern is adjusted to more accurately reflect the training pattern. Also, ANs in the neighbourhood of the winning AN are proportionately adjusted to reflect the training pattern. The learning process is iterative and continues until a good enough map has been found. The *quantisation error* is a measure of the goodness of the map and can be defined as the sum of *Euclidean distances* of all patterns to their corresponding winning ANs.The quantisation error is defined as follows:

$$E_Q = \sum_{l=1}^{|P|} \|\rho_l - \mathbf{w}_t\|^2$$

where ρ_l is the training pattern and $\mathbf{w}_t$ is the weight vector of the winning AN o_t.

The main advantage of a SOM is the easy visualisation and interpretation of clusters formed by the map. Map visualisation is achieved by computing the unified distance matrix, which expresses the distances between the codebook vectors of adjacent neurons or by using some colour scale representation. Large distances represent cluster boundaries and small distances represent clusters. SOMs can also be used for classification by labelling each AN according to the most likely outcome of that AN, and using the label as a classification for input patterns. Other applications of SOMs include prediction and interpolation [28].

2.4.5 Split-and-merge

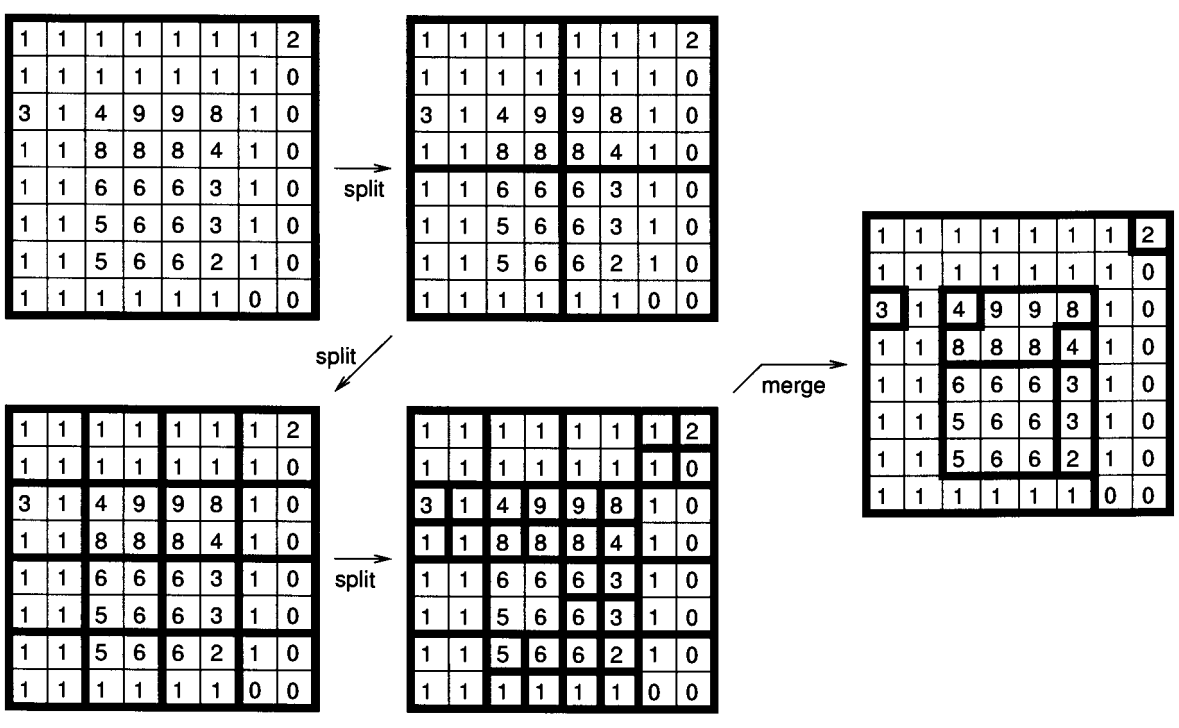

Figure 2.7: Split-and-merge illustration, $E \leq 1$

Split-and-merge is a hierarchical clustering algorithm that attempts to discover regions in a dataset, within which some property does not change abruptly [59]. Split-and-merge is mostly used for finding regions of homogeneous intensity in images and is important for robot vision. An image region is a set of connected pixels such that:

1. A region is *homogeneous*, *i.e.*:
   - The difference in intensity values of pixels in a region should be no more than some error E.
   - A polynomial surface of degree n can be fitted to the intensity values of pixels in the region with largest error less than E
2. For no two adjacent regions it is the case that the union of all the pixels in these two regions satisfies the homogeneous property.

Split-and-merge starts with one image region. If the region does not satisfy property 1 of the image region definition, then the image is partitioned into four image regions. If each of those image regions do not satisfy the homogeneous property, then each of the image regions is partitioned into four image regions. This process iterates until each image region is homogeneous. Once all splitting has been performed, the algorithm merges image regions until property 2 of the image region definition is satisfied. Figure 2.7 illustrates the split-and-merge algorithm with $E \leq 1$.

2.5 CONCLUSION

This chapter provided a taxonomy of the underlying methods employed by many data mining applications in use today. The paradigms of knowledge discovery, evolutionary computing, artificial neural networks and clustering were broadly discussed. This chapter provided the fundamental grounding for the algorithms and methods presented in the rest of this thesis. The next chapter discusses the GASOPE method as a means for obtaining structurally optimal polynomial expressions, as a means of performing function approximation.

Chapter 3

THE GASOPE METHOD

The previous chapter presented a number of machine learning paradigms used in this chapter, and throughout the rest of this thesis. This chapter presents a function approximation method that uses a genetic algorithm to evolve structurally optimal polynomial expressions (GASOPE). This genetic algorithm is one of the core aspects of the algorithm discussed in the next chapter.

3.1 INTRODUCTION

The study of function approximation can be broken up into two classes of problems. One class deals with a function being explicitly stated, where the objective is to find a computationally simpler type of function, such as a polynomial, that can be used to approximate a given function. The other class deals with finding the best function to represent a given set of data points (or patterns). The latter class of function approximation plays a very important role in the prediction of continuous-valued outcomes, *e.g.* subscriber growth forecasts, time-series modelling, *etc.* This chapter concentrates on methods to construct functions that accurately represent a series of data points, at minimal processing cost.

Traditional methods to perform this type of function approximation include the frequently used discrete least-squares method, regression, Taylor polynomials and Lagrange polynomials. Other methods include neural networks and some evolutionary algorithm paradigms.

This chapter develops a genetic algorithm approach to evolve structurally optimal polynomial expressions (GASOPE) in order to describe a given data set. A fast clustering algorithm

is used to reduce the pattern space and thereby reduce the training time of the algorithm. Highly specialised mutation and crossover operators are used to directly optimise the polynomial expressions, and to exploit similarities between the various polynomial expressions in the search space.

The remainder of this chapter is organised as follows: Section 3.2 presents an overview of various function approximation techniques. The implementation of the genetic algorithm polynomial approximator is presented in detail in section 3.3. The section presents the clustering algorithm used to cluster the training data, and introduces the representation, specialised mutation and crossover operators, and the hall-of-fame, all of which ensure the structural optimality of the evolved polynomials. The experimental procedure, data sets and results are discussed in section 3.4. Finally, section 3.5 presents the summarised findings and envisioned future developments to the method.

3.2 THEORY OF FUNCTION APPROXIMATION

As was mentioned earlier, function approximation can be broken up into two classes of problems: One class dealing with the simplification of a defined function in order to determine approximate values for that defined function, and the other class dealing with finding a function that best describes a set of data points. This section discusses, in detail, traditional methods to perform both classes of function approximation, such as the discrete least squares approximation, regression, Taylor polynomials and Lagrange Polynomials. This section also discusses other approaches, such as neural networks and evolutionary computing.

3.2.1 Discrete least squares approximation

The following is a brief adaptation of Fraleigh and Beauregard [40], and Burden and Faires [16]. The method of least squares involves determining the best linear approximation to an arbitrary set of $m = |P|$ data points $\{(a_1, b_1), \cdots, (a_m, b_m)\}$, by minimising the least squares error:

$$E_{SS} = \sum_{i=1}^{m} [b_i - b_i^{\bullet}]^2$$

where $b_i^\bullet$ represents the predicted output of some arbitrary function, *e.g.*

$$\begin{aligned} b_i^\bullet &= r_0 + r_1 a_i + \cdots + r_{n-1} a_i^{n-1} + r_n a_i^n \\ &= \sum_{j=0}^{n} r_j a_i^j \end{aligned} \tag{3.1}$$

where $n+1$ represents the maximum number of terms.

The coefficients r_0 through r_n of the polynomial function mentioned above, can be determined by solving the linear system:

$$\mathbf{b} \approx A\mathbf{r} \tag{3.2}$$

where $\mathbf{r}^T = [\ r_0 \quad r_1 \quad \cdots \quad r_n\ ]$, which means obtaining the least-squares solution by solving the overdetermined linear system:

$$(A^T A)\mathbf{r} = A^T \mathbf{b} \tag{3.3}$$

to which there is probably no exact solution. Matrix A is of the form:

$$A = \begin{bmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^n \\ 1 & a_2 & a_2^2 & \cdots & a_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & a_m & a_m^2 & \cdots & a_m^n \end{bmatrix} \tag{3.4}$$

and vector $\mathbf{b}^T = [\ b_0 \quad b_1 \quad \cdots \quad b_n\ ]$.

Obviously, the above method requires a decision as to what function to use to calculate the least-squares fit. Many types of functions can be fitted, *e.g.* polynomial, exponential, logarithmic, *etc.* In the simple polynomial case, however, at least a decision needs to be taken as to the value of n. Because a least-squares fit is empirically obtained from a set of data points, the interpolation characteristics of such a fit are reasonably good. However, for the same reason, a least-squares fit has poor extrapolation properties, particularly when an extrapolated point lies far away from the data set.

The least squares method is relatively fast, because it only requires the reduction of one linear system in order to obtain the best approximation of the problem space for a given architecture (approximating function). However, outliers can exercise considerable influence on the r_j coefficients and can lead to poor generalisation.

3.2.2 Regression

Regression can be described as the process of discovering the most plausible and most easily understandable relationship between a set of independent variables and a dependant variable [96]. A dependent variable and an independent variable are *correlated* when there is a relationship between them, *i.e.* an increase in the independent variable results in a decrease in the dependent variable or an increase in the independent variable results in an increase in the dependent variable. However, it is important to note that correlation between variables does not imply causality, *i.e.* if there is a relationship between an independent and a dependent variable, it is not necessarily the case that changes in the dependent variable are directly caused by the independent variable in the real-world.

The *coefficient of determination* R^2 provides an indication of how well a discrete least squares approximation (model) of section 3.2.1 fits the observed data. The coefficient of determination R^2 is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(b_i - b_i^{\bullet})^2}{\sum_{i=1}^{m}(b_i - \overline{b})^2} \tag{3.5}$$

where m is the size of the training set, b_i is the target output of pattern i, $\overline{b}$ is the mean of the target values and $b_i^{\bullet}$ is the predicted output of pattern i. The coefficient of determination has the range $[0, 1]$, where values closer to 1 indicate a strong correlation between the data set and the model and values closer to 0 indicate no relation between the data set and the model. The *correlation coefficient* is defined as $R = \sqrt{R^2}$ for simple linear regression, where there is only one independent variable.

The *adjusted coefficient of determination* R_a^2 is used as an indication of how well a discrete least squares approximation fits the observed data, factoring in the complexity of the discrete least squares approximation. The adjusted coefficient of determination R_a^2 is defined as:

$$R_a^2 = 1 - \frac{\sum_{i=1}^{m}(b_i - b_i^{\bullet})^2}{\sum_{i=1}^{m}(b_i - \overline{b})^2} \cdot \frac{m-1}{m-k} \tag{3.6}$$

where k is the number of coefficients (free variables) of the least squares approximation and all other variables are defined as before.

Essentially, the adjusted coefficient of determination R_a^2 penalises fits that have larger numbers of free variables, *i.e.* two models may have the same or similar accuracy, but the model with the smaller number of free variables will be preferred. Thus, the adjusted coeffi-

cient of determination attempts to maximise the correlation between a model and the data set, while minimising the architecture of the model.

3.2.3 Taylor polynomials

The following is a brief adaptation of Haggarty [50]. Without loss of generality, assume x is a scalar. If f is an n-times differentiable function at a point a, then the Taylor polynomial of degree n for f at a is defined by:

$$T_{n,a}f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

The importance of Taylor polynomials is that they only involve simple addition and multiplication. Moreover, given x to any specified degree of accuracy, it is straightforward to evaluate such polynomial expressions to a comparable degree of accuracy.

Taylor's theorem provides an important result: Let f be $(n+1)$-times continuously differentiable on an open interval containing the points a and b. Then the difference between f and $T_{n,a}f$ at b is given by:

$$f(b) - T_{n,a}f(b) = \frac{(b-a)^{(n+1)}}{(n+1)!}f^{(n+1)}(c)$$

for some c between a and b. The error in approximating $f(x)$ by the polynomial $T_{n,a}f(x)$ is the term to the right of the equality in the above. The error at a point between the Taylor polynomial $T_{n,a}f(x)$ and any function f can be determined to any degree of accuracy. This, in turn, means that Taylor polynomials can be used to approximate any n times differential, continuous function at any specific point on that curve.

3.2.4 Lagrange polynomials

Taylor polynomials are not appropriate for interpolation [16]. The n^{th} Lagrange interpolating polynomial is an alternative to Taylor polynomials that allows the approximation of a defined function over an interval [16]. The definition of an n^{th} Lagrange interpolating polynomial is as follows: If $x_0, x_1, \cdots, x_n$ are $n+1$ distinct numbers and f is a function whose values are given at these numbers, then there exists a unique polynomial P(x) of degree at most n with

the property that

$$f(x_k) = P(x_k)$$

for each $k = 0, 1, \cdots, n$. This polynomial is given by

$$P(x) = f(x_0)L_{n,0}(x) + \cdots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x) \tag{3.7}$$

where

$$\begin{aligned} L_{n,k} &= \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)} \\ &= \prod_{i=0,i\neq k}^{n} \frac{(x-x_i)}{(x_k-x_i)} \end{aligned} \tag{3.8}$$

for each $k = 0, 1, \cdots, n$. In order to fit an n-degree Lagrange polynomial, the method should be provided with $n+1$ control points that are uniformly distributed throughout the interval of the defined function.

It can be proved that the error between the Lagrange polynomial and the defined function is bounded over an interval. Suppose $x_0, x_1, \cdots, x_n$ are distinct numbers in the interval $[a,b]$ and $f \in C^{n+1}[a,b]$. Then, for each x in $[a,b]$, a number $E(x)$ in (a,b) exists with

$$f(x) = P(x) + \frac{f^{(n+1)}(E(x))}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n) \tag{3.9}$$

where $P(x)$ is the interpolating polynomial given by equation (3.7).

3.2.5 Selecting the correct approximating polynomial order

Any function with n turning points can be reasonably described by an order $(n+1)$-degree polynomial. Assume a continuous function f with n turning points $(p_1, \cdots, p_n)$, then the derivative of f must necessarily be 0 at those turning points. A polynomial expression with $f(x) = 0$ at all points $(p_1, \cdots, p_n)$ has the factorised form

$$f(x) = (x-p_1)(x-p_2)\cdots(x-p_n)$$

and has the simplified form

$$f(x) = r_n x^n + r_{n-1}x^{n-1} + \cdots + r_0 x^0$$

Integration of the simplified form yields

$$f(x) = \frac{r_n}{n}x^{n+1} + \frac{r_{n-1}}{n-1}x^n + \cdots + \frac{r_0}{1}x^1 + C$$

which is of degree $n+1$. Thus, if a polynomial is used to approximate a defined function or dataset, the degree of the polynomial approximation should always be selected as one more than the number of turning points of the original function or dataset. Higher order approximations can lead to over-fitting, because too many free variables (coefficients and terms) are available to any given polynomial approximation technique.

3.2.6 Artificial neural networks

Artificial neural networks with at least one hidden layer have been proved to be universal approximators [57], [58]. This means that a neural network can approximate any nonlinear mapping to a desired degree of accuracy, provided that enough hidden units are provided in the hidden layer. With reference to the previous section, Basson and Engelbrecht showed that for any function, the optimal number of hidden units that should be provided in the hidden layer should be one more than the number of turning points of the function [10]. However, this result assumes prior knowledge about the input space.

With reference to section 2.3, neural networks suffer from a number of problems when performing function approximation:

- The training of neural networks is computationally time consuming, especially when a large number of data points, are used and for large architectures.
- Finding the optimal architecture is crucial to ensure optimal interpolation (generalisation) performance. Architecture selection further adds to the complexity of training.
- Depending on the training algorithm used, neural networks are susceptible to local minima.
- Neural networks are sensitive to initial conditions and values of training parameters.
- While neural networks do have extrapolation capabilities, this deteriorates the further extrapolation points lie from the training set.

3.2.7 Evolutionary computing

A number of evolutionary computing approaches have been applied to function optimisation. Wilson describes a genetic algorithm that performs a piecewise-linear approximation to any arbitrary function [102]. His findings yielded arbitrary close approximations that were efficiently distributed over a function's domain.

Angeline, discusses a model that uses genetic programming to select a system of equations that are optimised in a neural network like fashion, in order to predict chaotic time-series [6]. His goal, specifically, was to evolve task specific activation functions for neural network-like systems of equations, *i.e.* activation functions that were not sigmoidal in nature.

Nikolaev and Iba discuss a genetic program that uses Chebishev Polynomials as building blocks for a tree-structured polynomial expression [76]. Their findings indicate that the tree-structured polynomial representation produced superior results on several benchmark and real-world time-series prediction problems, both in terms of training and generalisation accuracy.

With reference to section 2.2, evolutionary computing approaches have a number of factors that need to be considered before application:

- The training of evolutionary computing paradigms is computationally time consuming, especially when a large number of data points are used and for a large number of individuals.
- A suitable representation scheme must be chosen in order to adequately describe the problem space.
- A suitable fitness function must be selected in order to adequately describe the problem space.
- Depending on the problem, evolutionary computing paradigms could be sensitive to initial conditions and training parameters.
- Using a suitable chromosome representation scheme, the interpolation ability of an evolutionary computing algorithm can be engineered to be as good as any numerical analysis technique, particularly if the evolutionary computing algorithm utilises numerical methods. However, the evolutionary computing algorithm's extrapolation abilities will only be as good as the underlying numerical methods.

3.3 GASOPE STRUCTURE

This section discusses the implementation specifics of a genetic algorithm that evolves structurally optimal polynomial expressions (GASOPE) and heavily borrows from the ideas presented in section 3.2. Essentially, the algorithm is a three stage process that consists of:

1. a fast, rough k-means clustering algorithm to reduce the instance space,
2. a genetic algorithm to evolve polynomial expressions and
3. a hall-of-fame to find structurally optimal polynomial expressions.

Each of these sections will be discussed and elaborated on in full.

3.3.1 K-means clustering

One of the primary problems with all machine learning paradigms, is the need to iterate over each training pattern in order to calculate an error metric (in the case of a neural network) or to calculate the fitness (in the case of an evolutionary algorithm) of an individual. This is especially a problem with very large data sets. Special strategies have to be utilised, such as active learning for artificial neural networks 2.3.3, to solve the large data set problem.

Clustering has been used in order to try to break the aforementioned restriction. The idea is to perform a fast, rough clustering of the training data, and then to draw a stratified random sample from the clusters, to be used in a manner similar to that of Engelbrecht and Brits [32].

A stratified random sample [96], of size s, is drawn from k clusters of training patterns, where each cluster represents a stratum of homogeneous training patterns (according to some characteristic inherent in the data), instead of using all of the available training patterns. The stratified random sample is drawn proportionally from each of the k clusters, *i.e.*:

$$S = \cup_{\delta=1}^{k} C_{\delta}'$$

where

$$C_{\delta}' \subset C_{\delta} : |C_{\delta}'| = \frac{|C_{\delta}|s}{|P|}$$

and $|C_{\delta}|$ is the (stratum) size of cluster C_{δ} and $|P|$ is the size of the data set. Obviously, a proportional sample would be meaningless unless each strata was homogeneous with respect

to some characteristic. Also, if a cluster consists of just outliers, a proportional sample will prevent these outliers from skewing the data distribution.

The GASOPE method uses the k-means clustering algorithm (of section 2.4.2) to obtain the homogeneous strata mentioned above. The GASOPE method uses a simple heuristic in order to increase the performance of the k-means clustering algorithm, which requires the inclusion of a standard deviation measure for each cluster centroid. The algorithm proceeds as follows:

1. Initialise k centroids $(\mathbf{w}_1, \cdots, \mathbf{w}_k)$ such that each centroid is initialised to one input vector $\mathbf{w}_\delta = \rho_\delta, \delta \in \{1, \cdots, k\}$ and initialise k centroid deviation vectors $(\sigma_1, \cdots, \sigma_k)$ such that $\sigma_\delta = 0$, $\delta \in \{1, \cdots, k\}$, where each cluster C_δ is associated with the centroid $\mathbf{w}_\delta$ and the centroid deviation vector σ_δ.

2. For each input vector ρ_l, where $l \in \{1, \cdots, |P|\}$

   (a) If $(\rho_l \in C_\omega, \omega \in \{1, \cdots, k\}) \wedge ((\rho_l < \mathbf{w}_\omega - \sigma_\omega) \vee (\rho_l > \mathbf{w}_\omega + \sigma_\omega))$, then find the nearest centroid $\mathbf{w}_{\delta_*}$, *i.e.* if

   $$\|\rho_l - \mathbf{w}_{\delta *}\| \leq \|\rho_l - \mathbf{w}_\delta\|,$$

   $$(\delta = 1, \cdots, k) \wedge (\delta \neq \delta_*)$$

   then

   $$C_{\delta_*} = C_{\delta_*} \cup \{p_l\}$$

3. For each cluster C_δ, where $\delta \in \{1, \cdots, k\}$

   (a) Update the cluster centroid $\mathbf{w}_\delta$ to be the centroid of all the samples currently in C_δ so that

   $$\mathbf{w}_\delta = \frac{\sum_{\rho_l \in C_\delta} \rho_l}{|C_\delta|}$$

   (b) Update the cluster centroid deviation vector σ_δ to be the standard deviation of all the samples currently in C_δ so that

   $$\sigma_\delta = \sqrt{\frac{\sum_{\rho_l \in C_\delta} \rho_l^2 - \frac{(\sum_{\rho_l \in C_\delta} \rho_l)^2}{|C_\delta|}}{|C_\delta| - 1}}$$

4. Compute the error function

$$E_Q = \sum_{\delta=1}^{k} \sum_{\rho_l \in C_\delta} \|\rho_l - \mathbf{w}_\delta\|^2$$

5. Return to step 2 until E_Q does not change significantly or cluster membership does not change.

Essentially, the centroid $\mathbf{w}_\delta$ and the centroid standard deviation σ_δ in the above algorithm allow the k-means clustering algorithm to fit k hyper-cubes over an n-dimensional pattern space. Any pattern not within the bounds of the hyper-cube of the cluster of which the pattern is a member becomes eligible for selection in step (2a) of the algorithm. This pattern selection strategy drastically reduces the number of comparisons that need to be made for each training iteration of the algorithm, resulting in improved performance over the normal direct k-means clustering algorithm.

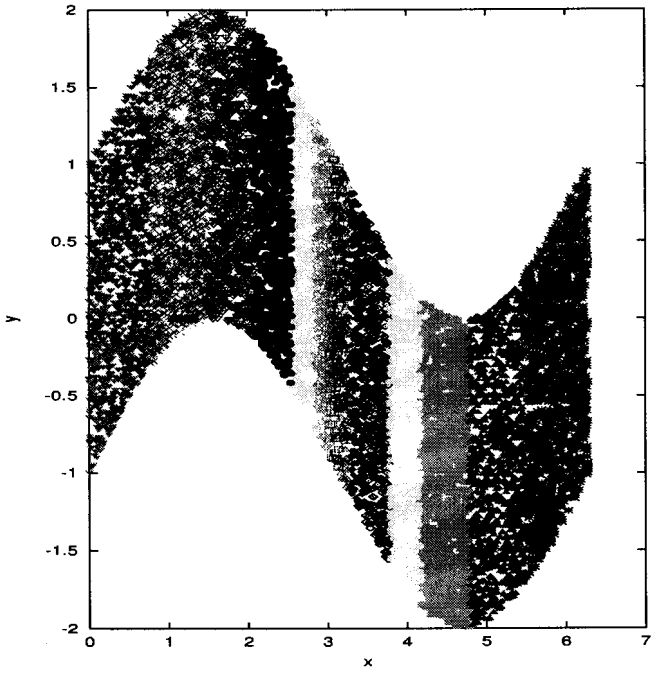

Figure 3.1: K-means output for $y = sin(x) + \mathrm{U}(-1,1), \mathrm{x} \in [0, 2\pi]$

Figure 3.1 demonstrates a typical output of the above k-means clustering algorithm for 15 cluster centroids. What is interesting to note, is that larger clusters form around the turning

points of the function. These larger clusters demonstrate that the information content of the function is greatest near the turning points of a function, where the derivative of the function is changing more rapidly. This, in turn, indicates that the efforts of the function approximation technique should be concentrated on the regions near the turning points of the function. Proportional sampling will select more patterns from larger clusters, and will therefore concentrate the efforts of the function approximation technique on the regions near the turning points. The idea is similar to an incremental learning approach used by Engelbrecht for artificial neural networks [35]

With regards to the bias-variance dilemma [44], clustering minimises the variance component of the GASOPE method. The genetic algorithm of the next section minimises the bias component of the GASOPE method.

3.3.2 Genetic algorithm for function approximation

The following section discusses the core algorithms employed by the genetic algorithm component of the GASOPE method. The *introduction* introduces the reader to the complexity of the technique used by the genetic algorithm. The *representation* of each individual in a population is then discussed. The *initialisation* values of each individual is presented. The *mutation* and *crossover* operators employed by the genetic algorithm are discussed. The fitness function employed by the genetic algorithm is discussed. Finally, the algorithm to guide the *genetic algorithm optimisation process* is presented.

Introduction

In sections 3.2.1 and 3.2.3 Taylor polynomials and discrete least squares approximation were discussed in terms of their relevance to function approximation. The definition of the linear function presented in equation (3.1) is extended to the non-linear form from:

$$b_i^{\bullet} = \sum_{j=0}^{n} r_j a_i^j$$

to:

$$b_i^{\bullet} = \sum_{\tau=0,\sum_{j=1}^{m}\lambda_j=\tau}^{n} \left(r_{(\lambda_1,\lambda_2,\cdots,\lambda_m)} \prod_{q=1}^{m} a_{i,q}^{\lambda_q} \right) \tag{3.10}$$

where m is the dimensionality of the input space, n is the maximum polynomial order, λ_q is the order of attribute $a_{i,q}$ and $r_{(\lambda_1,\lambda_2,\cdots,\lambda_m)}$ is a real-valued coefficient. This definition allows the representation of functions such as:

$$\begin{aligned} b_i^{\bullet} = \; & r_{(0,0)} + r_{(1,0)}a_{i,1} + r_{(0,1)}a_{i,2} + \\ & r_{(1,1)}a_{i,1}a_{i,2} + r_{(2,0)}a_{i,1}^2 + r_{(0,2)}a_{i,2}^2 \end{aligned}$$

for $m = 2$ and $n = 2$. If the value of the coefficients $r_{(\lambda_1,\lambda_2,\cdots,\lambda_m)}$ are efficiently determined using the least squares approximation (from equation (3.3)), then all the genetic algorithm is required to do is to algorithmically determine the optimal approximating polynomial structure. The definition of optimality used throughout this thesis is bimodal: both the smallest polynomial structure and the best possible function approximation are required. A simplistic approach would be to generate every possible combination of a function, and test the predicted result of the function against the data set. However, such an exhaustive search approach is prohibitive. This section does, however, continue to derive an upper bound on the genetic algorithm search space, should an exhaustive search be undertaken.

First, the total number of unique terms t generated by equation (3.10) is determined. Select, with repetition, p inputs from a set of m inputs. This problem is similar to determining the number of n-multi-sets of size p, where $p \in \{0,\cdots,n\}$. There are

$$t = \sum_{p=0}^{n} \left(\begin{array}{c} p+m-1 \\ p \end{array} \right) \tag{3.11}$$

such multi-sets (terms) [36]. By applying induction and Pascal's formula, equation (3.11) is simplified to

$$t = \left(\begin{array}{c} m+n \\ n \end{array} \right) \tag{3.12}$$

The number of function choices, u, is calculated by choosing, without repetition, q terms from the set of t terms:

$$u = \sum_{q=0}^{t} \left(\begin{array}{c} t \\ q \end{array} \right) \tag{3.13}$$

Using the Binomial theorem, equation (3.13) is simplified to:

$$u = 2^t \tag{3.14}$$

From equations (3.12) and (3.14), the number of terms and function choices can be determined. A 10-dimensional input space with a maximum polynomial order of 3, has 286 possible terms and 2^{286} function choices. To iteratively calculate and test each function choice against a set of S training patterns is thus computationally difficult. Genetic algorithms, however, can be used to determine solutions to such difficult problems, because genetic algorithms implement a highly parallel search.

Representation

The representation used by the algorithm is fairly simple and is, in fact, a representation of equation (3.10). Each individual is made up of a set I_ω of unique, term-coefficient mappings, *e.g.*

$$I_\omega = \{(t_0 \rightarrow r_0), \cdots, (t_{p-1} \rightarrow r_{p-1})\}$$

where p is the maximum set size (maximum number of terms) and r_ξ, for $\xi \in \{0, \cdots, p-1\}$, is a real-valued coefficient (re-mapped from equation (3.10) for the sake of simplicity). Each term t_ξ is made up of a set T_ξ of unique, variable-order mappings, *e.g.*

$$T_\xi = \{(a_{\xi,1} \rightarrow \lambda_{\xi,1}), \cdots, (a_{\xi,m} \rightarrow \lambda_{\xi,m})\}$$

where m is the number of inputs (variables), $a_{\xi,\tau}, \tau \in \{1, \cdots, m\}$ is an integer representing an input value and $\lambda_{\xi,\tau}$ is a natural-valued order. In practise, these two sets are maintained as a two-dimensional variable-length, sorted array, which allows only unique insertion.

Initialisation

Each individual I_ω in a population G_{GA} is initialised by randomly selecting variable-order pairs, in order to build a term T_ξ up to a maximum polynomial order $e \in \{0, \cdots, n\}$. This process is repeated until the number of terms equals the maximum number of terms p. The initialisation of an individual is fully described by the following pseudo-code algorithm:

1. Set $I_\omega = \{\}$
2. While $|I_\omega| < p$ do
   (a) Set $T_\xi = \{\}$

(b) Select $e \in \{0, \cdots, n\}$ uniformly from the maximum polynomial order n.

(c) While $e > 0$ do

i. Select $1 \leq f \leq e$ uniformly from the available orders e.

ii. Select $1 \leq g \leq m$ uniformly from the set of m inputs.

iii. If $|T_\xi| < |T_\xi \cup \{(g \to f)\}|$ then $e := e - f$, *i.e.* decrease the number of available orders e.

iv. Set $T_\xi = T_\xi \cup \{(g \to f)\}$

(d) Set $I_\omega = I_\omega \cup \{(T_\xi \to 0)\}$ as shown in figure 3.2

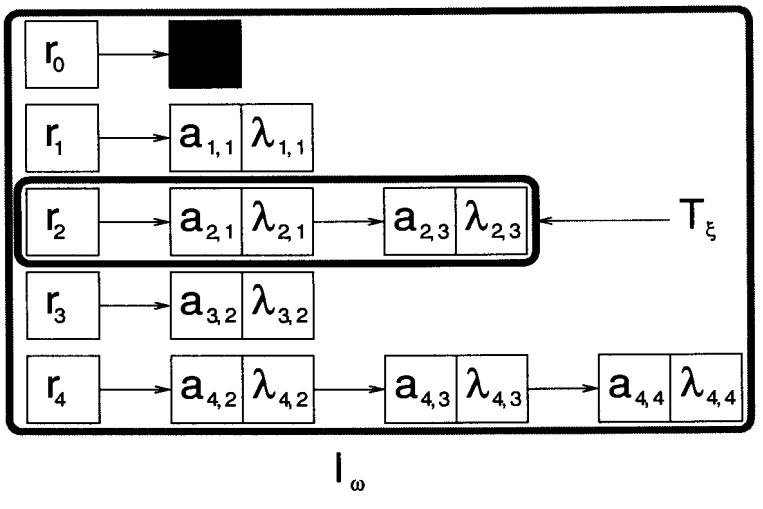

Figure 3.2: Illustration of GASOPE chromosome initialisation for an individual I_ω

Mutation operators

The mutation operators serve to inject new genetic material into a population of individuals, thus the mutation operator broadens the search space. Four mutation operators are used by the genetic algorithm, namely shrink, expand, perturb and reinitialise:

- **Shrink operator**: The shrink operator is fairly simple to implement and has the objective to remove, arbitrarily, one of the term-coefficient pairs T_ξ from the individual I_ω,

as indicated by the crossed out section in figure 3.3. The pseudo-code for the shrink operator is as follows:

1. Select $T_\xi \in I_\omega$ uniformly from the set of terms I_ω.
2. Set $I_\omega = I_\omega / \{T_\xi\}$ as shown in figure 3.3.

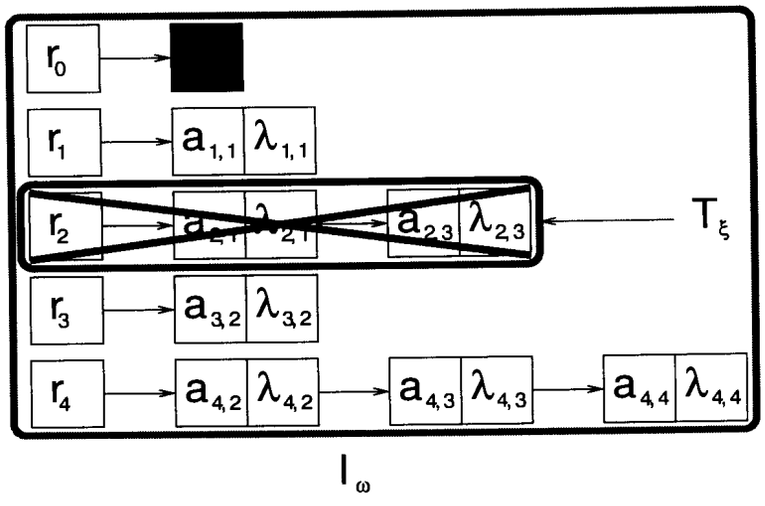

Figure 3.3: Illustration of the GASOPE shrink operator for an individual I_ω

- **Expand operator**: The expand operator adds a new random term-coefficient pair to the individual I_ω (only if $I_\omega < p$). The pseudo-code for the expand operator is as follows:

1. If $|I_\omega| < p$ then
   (a) Set $T_\xi = \{\}$
   (b) Select $e \in \{0, \cdots, n\}$ uniformly from the maximum polynomial order n.
   (c) While $e > 0$ do
      i. Select $f \in \{1, \cdots, e\}$ uniformly from the available orders e.
      ii. Select $g \in \{1, \cdots, m\}$ uniformly from the set of m inputs.
      iii. If $|T_\xi| < |T_\xi \cup \{(g \rightarrow f)\}|$ then $e := e - f$, *i.e.* decrease the number of available orders e.
      iv. Set $T_\xi = T_\xi \cup \{(g \rightarrow f)\}$

(d) Set $I_\omega = I_\omega \cup \{(T_\xi \rightarrow 0)\}$ as shown in figure 3.4.

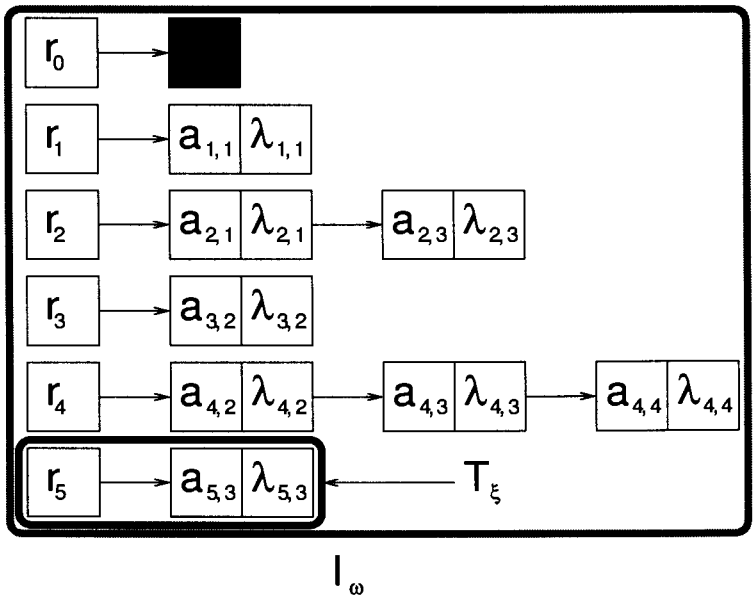

Figure 3.4: Illustration of the GASOPE expand operator for an individual I_ω

- **Perturb operator**: The perturb operator is fairly complicated and requires the algorithm to select a term T_ξ from the individual I_ω, and to adjust one of the variable-order mappings. This adjustment can either add, remove or adjust an order in a variable-order mapping and is applied uniformly (with equal probability) over all three actions. The pseudo-code for the perturb operator is as follows:

  1. Select $T_\xi \in I_\omega$ uniformly from the set of terms I_ω.
  2. Calculate the number of orders available

$$e := p - \sum_{\nu=1}^{|T_\xi|} \lambda_{I_\omega,\nu}.$$

  3. Select $g \in \{1, \cdots, m\}$ uniformly from the set of m inputs.
  4. Select $h \in \mathrm{U}(0,1)$ as a uniformly distributed random number.
  5. If $h < 0.333$ then

(a) Set $T_\xi = T_\xi / \{(g \rightarrow \lambda_g)\}$, *i.e.* remove the g^{th} variable-order mapping from set T_ξ as shown by the crossed out section in figure 3.5.

6. Else if $h < 0.666$ then

   (a) Select $f \in \{1, \cdots, e\}$ uniformly from the available orders e.

   (b) Set $T_\xi = T_\xi \cup \{(g \rightarrow f)\}$ as shown by the large box in figure3.5.

7. Else

   (a) Set $T_\xi = T_\xi / \{(g \rightarrow \lambda_g)\}$, *i.e.* remove the g^{th} variable-order mapping from set T_ξ.

   (b) Set $e := e + \lambda$, *i.e.* increase the number of available orders e.

   (c) Select $f \in \{1, \cdots, e\}$ uniformly from the available orders e as shown by the small box in figure 3.5.

   (d) Set $T_\xi = T_\xi \cup \{(g \rightarrow f)\}$

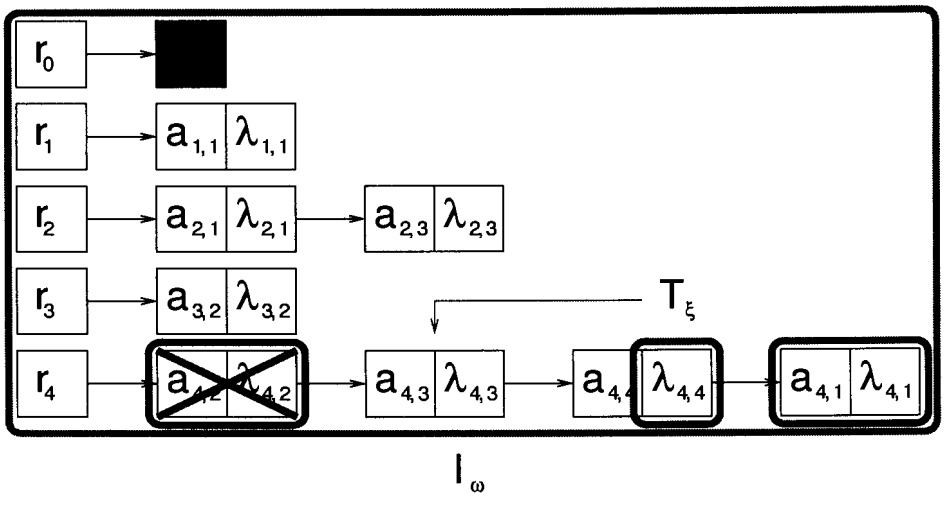

Figure 3.5: Illustration of the GASOPE perturb operator for an individual I_ω

- Finally, the **reinitialise operator** is just a re-invocation of the initialisation operator.

Crossover operator

The crossover operator serves to retain genetic material from one generation of individuals to the next, thus the crossover operator directs the search space toward a particular solution. The crossover operator used by the genetic algorithm selects a subset of two individuals in order to construct a new chromosome. Term-coefficient mappings are selected to construct the new chromosome at random, with a higher probability of selection given to term-coefficient mappings that are prevalent in both individuals. A ratio of 80:20 was used (shown below), because, on average, the new individual generated by these parameters was found to be roughly the same length as its longer parent. The pseudo algorithm for the crossover operator is as follows:

1. Let $I_\alpha = \{\}$ be a new term-coefficient set (individual) as shown in figure 3.6.
2. Let $I_\beta \in G$ be any term-coefficient set in the population of individuals G_{GA} as shown in figure 3.6.
3. Let $I_\gamma \in G$ be any term-coefficient set in the population of individuals G_{GA} as shown in figure 3.6.
4. Let $A = I_\beta \cap I_\gamma$ be the intersection of term-coefficient mappings as shown in figure 3.6.
5. Let $B = (I_\beta / I_\gamma) \cup (I_\gamma / I_\beta)$ be the union of the exclusions as shown in figure 3.6.
6. Set the temporary set iterator $e := 1$ for A.
7. While $e < |A|$ and $I_\alpha < p$ do
   (a) Select $h \in \mathrm{U}(0,1)$ as a uniformly distributed random number.
   (b) If $h < 0.8$ then $I_\alpha = I_\alpha \cup \{A_e\}$
   (c) Set $e := e + 1$
8. Set the temporary set iterator $e := 1$ for B.
9. While $e < |B|$ and $I_\alpha < p$ do
   (a) Select $h \in \mathrm{U}(0,1)$ as a uniformly distributed random number.

(b) If $h < 0.2$ then $I_\alpha = I_\alpha \cup \{B_e\}$

(c) Set $e := e + 1$

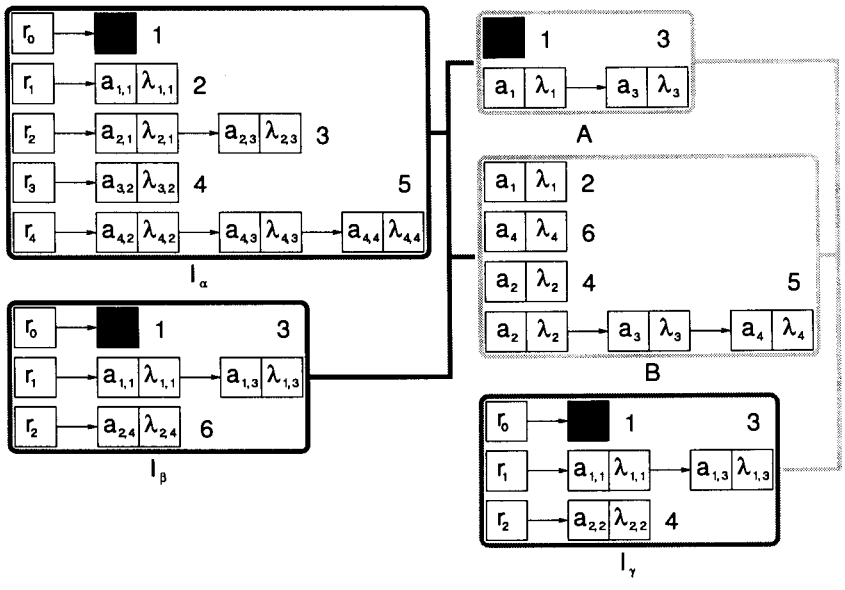

Figure 3.6: Illustration of the GASOPE crossover operator for individuals I_α, I_β and I_γ

Fitness function

The fitness function is an important aspect of a genetic algorithm, in that it serves to direct the algorithm toward optimal solutions. The fitness function used by the genetic algorithm is similar to the *adjusted coefficient of determination* (from equation (3.6)). The fitness function is defined as:

$$R_a^2 = 1 - \frac{\sum_{i=1}^{s}(b_i - b^{\bullet}_{I_\omega,i})^2}{\sum_{i=1}^{s}(b_i - \overline{b}_i)^2} \cdot \frac{s-1}{s-k} \tag{3.15}$$

where s is the sample size, b_i is the actual output of pattern i, $b'_{I_\omega,i}$ is the predicted output of individual I_ω for pattern i, and the model complexity d is calculated as follows:

$$d = \sum_{\xi=1}^{|I_\omega|} \sum_{\tau=1}^{|T_\xi|} \lambda_{\xi,\tau} \tag{3.16}$$

where I_ω is an individual in the set P of individuals, T_ξ is a term of I_ω and $\lambda_{\xi,\tau}$ is the order of term T_ξ. This fitness function penalises the complexity of an individual I_ω by penalising the number of multiplications needed to calculate the predicted output of that individual,*i.e.* the number of terms and their order.

In order to calculate the fitness of an individual, however, the algorithm requires the coefficients in the set I_ω to be calculated. Matrix A (from equation (3.4)) is populated with the combination of terms represented by each term-coefficient mapping, *e.g.* if the term is $a_0 a_1^2$, the algorithm multiplies out each of the input attributes for a particular pattern. Matrix A is thus populated from left to right with terms from each pattern in the sample space (where the patterns proceed from top to bottom). The vector $\mathbf{b}$ is made up of the target output for each pattern. After reducing the linear system shown by equation (3.3), vector $\mathbf{r}$ represents the coefficients of each of the term-coefficient mappings.

The genetic algorithm optimisation process

The GASOPE algorithm is summarised below:

1. Let $g = 0$ be the generation counter
2. Initialise a population $G_{GA,g}$ of N individuals, *i.e.*

$$G_{GA,g} = \{I_\omega | \omega = 1, \cdots, N\}$$

3. While $g < M$, where M is the total number of generations, do
   (a) Sample $S \subset \cup_{\delta=1}^{k} C_\delta$ where C_δ is a cluster (stratum) of patterns
   (b) Determine the coefficients of each of the term-coefficient mappings in an individual I_ω by reducing $\mathbf{b} \approx A\mathbf{r}$ in terms of the sample S
   (c) Evaluate the fitness $R_a^2(I_\omega)$ of each individual in population $G_{GA,g}$ using the patterns in S
   (d) Let $G'_{GA,g} \subset G_{GA,g}$ be the top $x\%$ of the individuals to be involved in elitism
   (e) Install the members of $G'_{GA,g}$ into $G_{GA,g+1}$
   (f) Let $G''_{GA,g} \subset G_{GA,g}$ be the top $n\%$ of the individuals to be involved in crossover

(g) Perform crossover:

i. Randomly select two individuals I_α and I_β from $G''_{GA,g}$

ii. Produce offspring I_γ from I_α and I_β

iii. Install I_γ into $G'''_{GA,g}$

(h) Perform mutation:

i. Select an individual I_ω from $G'''_{GA,g}$

ii. Mutate I_ω

iii. Install I_ω into $G_{GA,g+1}$

(i) Evolve the next generation $g := g + 1$

It is important to note that each generation in the above algorithm draws a new stratified random sample from the training set. The entire process, thus, is not based on just one sample of the training set.

3.3.3 Hall-of-fame

This section discusses the need for a hall-of-fame. The hall-of-fame works like the hall-of-fame in classic arcade games, where the player that achieved a better score than any of the players in the hall-of-fame takes his/her rightful place (by entering his/her initials) and knocks the worst score off the list. For GASOPE, the hall-of-fame is essentially a set of unique, individual solutions, ranked according to their fitness value. After every generation the best individual is given the opportunity to enter the hall-of-fame. Entry into the hall-of-fame is determined as follows:

- If the best individual of a generation is structurally equivalent to an individual in the hall-of-fame, the fitness values of the individual in the hall-of-fame and the best individual are compared. The individual with the best fitness then replaces the individual in the hall-of-fame.

- Otherwise, if the best individual of a generation is not structurally equivalent to any individual in the hall-of-fame, the best individual is inserted relative to its fitness (or possibly not at all if its fitness is worse than any of the individuals in the hall-of-fame).

The hall-of-fame is not an elitism method; the individuals in the hall-of-fame do not further participate in the evolutionary process. The hall-of-fame simply keeps track of the best solutions for any given architecture. In the end the solution taken is not necessarily the best solution of the last generation, but the best over all generations. Ultimately, the purpose of the hall-of-fame is to ensure that the best, general solution is selected as the solution to the optimisation process.

Because the genetic algorithm of section 3.3.2 works with only a sample of the available patterns, certain function fits may not represent the true nature of the data set, particularly when such a data set is extremely noisy, because of the new sample used at each generation. For example, with a particularly poor sample selection from a noisy data set, the genetic algorithm may decide that a straight line is the optimal fit for the data set, when, in fact, a cubic function would have performed better on the whole. Following that, the genetic algorithm may decide that such a fit was the best fit seen so far in the optimisation process and will decide to retain that solution, ultimately leading to sub-optimal convergence. The hall-of-fame prevents this scenario from happening, because all best solutions compete for a place in the hall-of-fame. At the end of the optimisation process, the solutions in the hall-of-fame are tested against a validation set (a subset of the patterns withheld from training), to determine the ultimate solution.

3.4 EXPERIMENTAL RESULTS

This section discusses the experimental procedure and results of the GASOPE method *vs.* an artificial neural network, applied to various data sets generated from a range of generating functions. Section 3.4.1 presents the generating functions of the various data sets. The experimental procedure and program initialisation are explained in section 3.4.2. Section 3.4.3 presents the experimental results for the functions listed in section 3.4.1 and discusses the findings.

3.4.1 Functions

Table 3.1 presents a range of functions, f1 to f5, used to test the GASOPE algorithm. Some of these functions are illustrated by figures 3.7 to 3.9. These functions are all continuous over

Table 3.1: Function definitions

Name	Function
f1	$f(x_0) = sin(x_0) + \mathrm{U}(-1,1); \mathrm{x}_0 \in [0, 2\pi]$
f2	$f(x_0) = sin(x_0) + cos(x_0) + \mathrm{U}(-1,1); \mathrm{x}_0 \in [0, 2\pi]$
f3	$f(x_0) = x_0^5 - 5x_0^3 + 4x_0 + \mathrm{U}(-1,1); \mathrm{x}_0 \in [-2, 2]$
f4	$f(x_0, x_1) = sin(x_0) + sin(x_1) + \mathrm{U}(-1,1); \{\mathrm{x}_0, \mathrm{x}_1\} \in [0, 2\pi]$
f5	$f(x_0, x_1) = x_0^5 - 5x_0^3 + 4x_0 + x_1^5 - 5x_1^3 + 4x_1 + \mathrm{U}(-1,1); \mathrm{x}_0, \mathrm{x}_1 \in [-2, 2]$

their domains and have been injected with noise to illustrate the characteristics of the GASOPE method on noisy data sets. Additionally, the method has been tested on a number of interesting chaotic time-series problems:

- **Logistic map**: The first, and most basic, chaotic time-series problem is the Logistic map, whose generating function can be described in the following manner:

$$\frac{dx}{dt} = a \cdot x_n(1 - x_n)$$

  where $a = 4$ and $x_0 = 0.2$.

- **Henon map**: The next chaotic time-series problem is the Henon map, which can be described in the following manner:

$$\frac{dx}{dt} = 1 - a \cdot x_n^2 + b \cdot y_n$$

$$\frac{dy}{dt} = \frac{dx}{dt}$$

  where $a = 1.4, b = 0.3, x(0) \sim \mathrm{U}(-1,1)$ and $y(0) \sim \mathrm{U}(-1,1)$. The henon map is illustrated by figure 3.10.

- **Rossler attractor**: The next chaotic time-series problem is the Rossler attractor, whose generating function is:

$$\frac{dx}{dt} = -y_n - z_n$$

$$\frac{dy}{dt} = x_n - a \cdot y_n$$

$$\frac{dz}{dt} = b + z_n(x_n - c)$$

where $a = 0.2, b = 0.2, c = 5.7, x_0 = 1.0, y_0 = 0$ and $z_0 = 0$. This function should be generated using the Runga-Kutta order 4 method [16]. The Rossler attractor is illustrated in figures 3.11 to 3.14.

- **Lorenz attractor**: The last chaotic time-series problem is the Lorenz attractor, whose generating function is:

$$\frac{dx}{dt} = \sigma(y_n - x_n)$$

$$\frac{dy}{dt} = r \cdot x_n - y_n - x_n z_n$$

$$\frac{dz}{dt} = x_n y_n - b \cdot z_n$$

where $\sigma = 10, r = 28, b = 8/3, x_0 = 1.0, y_0 = 0$ and $z_0 = 0$. Once again, this function should be generated using the Runga-Kutta order 4 method [16]. The Lorenz attractor is illustrated in figures 3.15 to 3.18.

3.4.2 Experimental procedure

Each of the generating functions listed in section 3.4.1 were used to create a corresponding data set consisting of 12000 patterns. Each pattern consisted of the inputs and target outputs for the specific generating function, *e.g.* for the Rossler attractor, each pattern consisted of the three input components and one target output component. Each data set was scaled to create another data set, which consisted of scaled input components (to the range [-1,1]), to be used by a neural network for function approximation.

The neural network implementation used for comparison with the GASOPE method was trained until the maximum number of epochs were exceeded, using stochastic gradient descent. Only the hidden layer of the neural network used sigmoidal activation functions; the other layers used linear activation functions. The use of linear activation functions in specifically the output layers negates the need for the scaling of the neural network outputs. The neural network was initialised with an initial learning rate of 0.15 (a linearly decreasing learning rate was used), a momentum of 0.9 and a maximum number of epochs of 500. No checks were used to determine whether the neural network over-fitted the data.

Table 3.2: GASOPE initialisation

Variable	Value
Clusters	15
ClusterEpochs	10
FunctionMutationRate	0.1
FunctionCrossoverRate	0.2
FunctionGenerations	100
FunctionIndividuals	30
FunctionPercentageSampleSize	0.01
FunctionMaximumComponents	20
FunctionElite	0.1
FunctionCutOff	0.001

The GASOPE method was initialised using the values shown in table 3.2. The number of clusters ("Clusters") parameter sets the number of clusters to be used in the k-means optimisation process. A larger value results in increased training time due to the increase in comparisons between patterns and cluster centroid vectors. A smaller value may have implications for accuracy in that, strictly speaking, the number of cluster centroids should be at least as large as the number turning points of the underlying function that describes the data points. The number of clusters value was chosen as 15 because the value is larger than the number of turning points used by any of the above defined functions.

The number of cluster epochs ("ClusterEpochs") determines the number of times the training set is presented to the k-means clustering algorithm. A small number of cluster epochs results in rough cluster membership, whereas a large number of cluster epochs result in crisp cluster membership. However, the larger the number of cluster epochs, the longer the genetic algorithm takes to optimise. A value of 10 was chosen because the GASOPE method requires speed more than precision.

The linear system $(A^TA)\mathbf{r} = A^T\mathbf{b}$ (refer to equation (3.3)) is consistent. This means that whether the linear system $\mathbf{b} \approx A\mathbf{r}$ is overdetermined or underdetermined, the first linear system will still be reducible. However, the first linear system could potentially return a poor least

squares approximation if *Gauss-Jordan reduction* is used to solve the linear system. Gauss-Jordan reduction can introduce discontinuities into the reduction of the linear system when the data points used to populate that linear system lie too close to one another or a pivot in the system lies close to 0. These discontinuities typically occur when a value in the matrix is less than the minimum granularity of the floating point representation of the target platform.

An alternative method of matrix reduction that does not introduce discontinuities is *singular-value decomposition* [80]. Singular-value decomposition can always be performed, no matter how singular (non-invertible) a matrix is. Although both Gauss-Jordan reduction and singular-value decomposition have a maximum complexity of $O(n^3)$, singular-value decomposition requires the execution of more depth-3 nested loops than Gauss-Jordan reduction and is therefore slower than Gauss-Jordan reduction.

An alternative way to solve the poor least squares approximation problem, as outlined above, is based on data point clustering. The goal of clustering (from section 2.4) is to find homogeneous strata in a set of data points. A pattern in one cluster is dissimilar to all patterns in all other clusters. If we sample the training set by selecting at least one pattern from each cluster, the patterns should be dissimilar enough to return a good least squares approximation. The sample size ("FunctionPercentageSampleSize") parameter should thus be chosen large enough to include at least one pattern from each cluster. A large sample size results in a slower optimisation process because it directly increases the number of patterns to be presented to the GASOPE optimisation algorithm.

The accuracy of the GASOPE method depends both on the sample size and the number of clusters. An increase in the number of clusters leads to patterns being selected more uniformly throughout the domain of the search space. An increase in the sample size results in more patterns being selected from the turning points of the search space. The implication is that both the sample size and the number of clusters are equally important factors in ensuring the accuracy of the generated solutions.

The function cutoff ("FunctionCutOff") parameter controls a heuristic that prevents terms from appearing in a polynomial expression when that term's coefficients tend to 0. This cutoff results in a reduced set of terms and is thus used to prune each individual solution. The cutoff is applied after the calculation of each term's coefficient by checking whether the coefficient lies between the bounds of $[-FunctionCutOff, FunctionCutOff]$. If this is the case, the term

is removed from the polynomial expression. The other genetic algorithm parameters are fairly self explanatory and suffer from the problems discussed in section 2.2.4.

Both the neural network and the GASOPE method made use of three distinct sets of patterns. For each problem the 12000 data patterns were split up into a training set of 10000 patterns, a validation set of 1000 patterns and a generalisation set of 1000 patterns. The purpose of the training set is to train the two methods; the fitness function of the genetic algorithm and the forward- and back-propagation phase of the neural network use the training set as the driving force of their algorithms. The validation set is used to validate the interpolation ability of the genetic algorithm; the genetic algorithm uses the validation set to select the best individual from the hall-of-fame. The generalisation set is used to compare the two algorithms on unseen data patterns *i.e.*, their generalisation ability.

The results for each of the functions listed in section 3.4.1 were obtained by running 100 simulations of the corresponding data sets. Note though, that before each simulation was run, the data patterns were shuffled randomly among the three sets (training, validation and generalisation) and presented in this form to both the neural network and the GASOPE method. Results reported are averages over the 100 simulations together with standard deviations.

Table 3.3: GASOPE *vs.* NN pattern presentations

Method	Simulations	Epochs/ Generations	Individuals	Training patterns	Pattern presentations
GASOPE	100	100	30	0.01×10000	30000000
NN	100	500	1	10000	500000000

One criticism of comparing a neural network with the GASOPE method revolves around the complexity of each method. Using a training set of 10000 patterns, table 3.3 shows the calculated number of pattern presentations per method. Because the GASOPE method only samples 100 training patterns per generation ($FunctionPercentageSampleSize \times SampleSize = 0.01 \times 10000 = 100$), the number of pattern presentations that have to be performed are considerably reduced compared to that of the neural network. If the number of training patterns had not been included in this complexity measure, many would have argued that the experimental results are unfairly weighted against the neural network. Bearing in mind that the neural

network sees more of the training data during optimisation, this is not the case. Also, it is expected that the NN optimisation process will take on average 16.667 times longer than the GASOPE method to complete.

The breakdown of the total execution time of the GASOPE method for an average simulation run is as follows:

1. Approximately 16% of the time is spent on k-means clustering.
2. Approximately 42% of the time is spent on performing Gauss-Jordan matrix reductions.
3. Approximately 30% of the time is spent on performing pattern presentations.
4. Approximately 10% of the time is spent on solution and sample maintenance.

3.4.3 Results

This section discusses the experimental results of both the neural network mentioned in section 3.4.2 and the genetic algorithm presented in this chapter for the chosen function approximation tasks. The section is divided into three categories namely, noiseless data sets, noisy data sets and polynomial structure.

Noiseless data sets

Table 3.4 summarises the results of the noiseless application of the Henon map, Logistic map, Rossler attractor and the Lorenz attractor, respectively. All experiments utilising these functions used the GASOPE method with a maximum polynomial order of 3 and a neural network (NN) with a hidden layer size of 3, *i.e.* there were 3 hidden units. As mentioned in section 3.4.2, all other initialisation values were set according to table 3.2 and the initialisation criteria specified in section 3.4.2.

For each of the experiments shown in table 3.4, the GASOPE method performed significantly better, on average, than the neural network. This improvement was both in terms of training and generalisation accuracy and in terms of the average simulation completion time for each simulation run. What is interesting to note, is that the GA performed better than the NN on a range of chaotic time-series problems. Also, the GASOPE method is more robust than

Table 3.4: Comparison of GASOPE and NN on noiseless data (TMSE = mean squared error on training set, GMSE = mean squared error on test set (generalisation), σ indicates the standard deviation, $\bar{t}$ is average simulation completion time in seconds)

Function	GASOPE NN	*TMSE* *TMSE*	σ_{TMSE} σ_{TMSE}	*GMSE* *GMSE*	σ_{GMSE} σ_{GMSE}	$\bar{t}$ $\bar{t}$
Henon	GASOPE	0.000000	0.000000	0.000000	0.000000	0.83s
	NN	0.001258	0.011678	0.001266	0.011730	29.46s
Logistic	GASOPE	0.000000	0.000000	0.000000	0.000000	0.66s
	NN	0.055384	0.061578	0.055628	0.061842	20.50s
Rossler	GASOPE	0.000000	0.000000	0.000000	0.000000	0.91s
x component	NN	0.000754	0.000562	0.002488	0.002444	32.16s
Rossler	GASOPE	0.000000	0.000000	0.000000	0.000000	0.87s
y component	NN	0.000454	0.000348	0.001416	0.001640	32.15s
Rossler	GASOPE	0.000004	0.000000	0.000004	0.000000	0.90s
z component	NN	0.000388	0.000054	0.018146	0.041082	32.10s
Lorenz	GASOPE	0.000434	0.000004	0.000432	0.000028	0.88s
x component	NN	0.002064	0.001108	0.655078	0.634666	32.50s
Lorenz	GASOPE	0.000900	0.000014	0.000900	0.000086	0.95s
y component	NN	0.087084	0.034538	2.588240	2.377820	32.51s
Lorenz	GASOPE	0.000404	0.000028	0.000396	0.000050	0.96s
z component	NN	0.172464	0.193974	2.464340	2.326490	32.54s

the NN, because the standard deviations are smaller. Additionally, the NN average simulation completion time was more than 16.667 times the average completion time of the GASOPE method. Thus, the GASOPE method is orders of magnitude faster than the NN. Figures 3.10 to 3.18 show the function plots of most of the functions used in this section.

Noisy data sets

Table 3.5 represents the experimental results for the GASOPE method and the neural network (NN) as applied to a number of noisy data sets.

- **f1**: Function f1 represents the results of a noisy application of $\sin(x)$ over a domain of one period. The NN was trained using 3 hidden units in the hidden layer and the GASOPE method was trained with a maximum polynomial order of 3. All other initialisation parameters were set as shown by table 3.2 and specified in section 3.4.2. The GASOPE method performed slightly better than the NN in terms of training and generalisation error, and performed substantially better than the NN in terms of the average simulation completion time. A plot of the best GASOPE and NN output, evaluated according to generalisation ability of each end state of each simulation run, against the plot of the generalisation data set is shown in figure 3.7.

- **f2**: Function f2 represents the results of a noisy application of $\sin(x) + \cos(x)$ over a domain of one period. The NN was, once again, trained using 3 hidden units and the GASOPE method was trained with a maximum polynomial order of 3. The NN, in this case, performed slightly better than the GASOPE method in terms of accuracy. The GASOPE method, however, was still reasonably competitive against the NN. The GASOPE method performed substantially better than the NN in terms of the average simulation completion time. Figure 3.8 shows a plot of the best NN and GASOPE output for the function f2 against a plot of the generalisation data set.

- **f3**: Function f3 represents the results of a noisy application of a fifth order polynomial over the interval $[-2, 2]$. The GASOPE method used a maximum polynomial order of 5 and the NN used 5 hidden units. The GASOPE method performed substantially better than the NN, both in terms of accuracy and the average simulation completion time. A plot of the best GA and NN output against the plot of the generalisation data set is shown in figure 3.9.

- **f4**: Function f4 represents the results of a 2-dimensional application of function f1. The GASOPE method used a maximum polynomial order of 3 and the NN used 6 hidden units (2 dimensions, 2 turning points in each dimension). The NN performed slightly better than the GA in terms of training and generalisation accuracy, but performed significantly worse than the GASOPE method in terms of the average simulation completion time.

- **f5**: Function f5 represents the results of a 2-dimensional application of function f3. The GA used a maximum polynomial order of 5 and the NN used 10 hidden units (2 dimensions, 4 turning points in each dimension). The GASOPE method performed significantly better than the NN both in terms of training and generalisation accuracy, and in terms of the average simulation completion time.

- **Henon**: The Henon function represents the results of a noisy application of the Henon map. Uniformly distributed noise in the range $[-1,1]$ was injected into the output component of the data set. The GASOPE method used a maximum polynomial order of 3 and the NN used 3 hidden units. The NN performed slightly better than the GASOPE method in terms of training and generalisation accuracy, but performed significantly worse than the GASOPE method in terms of the average simulation completion time. Figure 3.10 shows a plot of the Henon map.

- **Lorenz**: The Lorenz functions represent the results of a noisy application of the Lorenz attractor in all three components. Uniformly distributed noise in the range $[-10,10]$ was injected into the each of the input and output components of the data set. The GASOPE method used a maximum polynomial order of 3 and the NN used 3 hidden units. For all three experiments of the Lorenz attractor, the GASOPE method performed significantly better than the NN both in terms of training and generalisation accuracy, and in terms of the average simulation completion time. Figures 3.15 to 3.18 show the plots for the Lorenz attractor.

Once again, it is interesting to note that the NN's average simulation completion time was more than 16.667 times the average simulation completion time of the GASOPE method. This showed that the GASOPE method is orders of magnitude faster that the NN.

Polynomial structure

This section discusses the structural optimisation ability of the GASOPE method.

Using Lagrange interpolating polynomials (section 3.2.4) a defined function can be approximated to any required degree of accuracy. If the function

$$y = xe^x, x \in [0,1.5]$$

Table 3.5: Comparison of GASOPE and NN on noisy data (TMSE = mean squared error on training set, GMSE = mean squared error on test set (generalisation), σ indicates the standard deviation, $\bar{t}$ is average simulation completion time in seconds)

Function	GASOPE NN	*TMSE* *TMSE*	σ_{TMSE} σ_{TMSE}	*GMSE* *GMSE*	σ_{GMSE} σ_{GMSE}	$\bar{t}$ $\bar{t}$
f1	GASOPE	0.339252	0.001538	0.339346	0.010278	0.75s
	NN	0.341698	0.003444	0.341108	0.009948	19.54s
f2	GASOPE	0.387740	0.002300	0.386500	0.011378	0.75s
	NN	0.341504	0.002420	0.340432	0.009524	19.87s
f3	GASOPE	0.337418	0.001496	0.336534	0.008804	0.78s
	NN	0.446104	0.034328	0.446128	0.037590	29.35s
f4	GASOPE	0.351866	0.002412	0.351748	0.010270	1.25s
	NN	0.342220	0.003066	0.343354	0.009826	47.36s
f5	GASOPE	0.336980	0.001826	0.337030	0.008900	1.36s
	NN	0.437576	0.090354	0.440616	0.085214	72.06s
Henon	GASOPE	0.745516	0.021636	0.748390	0.034662	0.80s
	NN	0.724794	0.027760	0.729226	0.038624	29.58s
Lorenz	GASOPE	46.663200	0.576780	46.760000	1.665810	0.93s
x component	NN	49.093600	1.640780	51.442600	2.628370	32.10s
Lorenz	GASOPE	52.554100	1.200120	53.048800	2.334640	0.87s
y component	NN	56.833200	2.887430	59.376100	4.060700	32.46s
Lorenz	GASOPE	55.493400	0.958328	55.562000	2.087360	0.90s
z component	NN	59.959000	2.625930	62.317100	3.420240	32.43s

is approximated with a Lagrange polynomial of degree 3 (with interpolation points $x = 0, x = 0.5, x = 1, x = 1.5$), the approximation

$$y = 1.3875x^3 + 0.057570x^2 + 1.2730x$$

is obtained. Using the GASOPE method, the result over 100 simulations (initialised using table 3.2 and a maximum polynomial order of 3) is consistently

$$y = 1.38552x^3 + 1.35944x - 0.0280584$$

The difference in MSE between these two methods is 0.000174, with the Lagrange polynomial being the more accurate of the two. The GASOPE method decided to substitute the x^2 term for the constant -0.0280584 (one less multiplication in the simplified form used by the GASOPE method) because the loss in accuracy was acceptable and it resulted in a smaller architecture.

Similarly, approximation of

$$y = sin(x), x \in [0, 2\pi]$$

with a Lagrange polynomial of degree 3 (with interpolation points $x = 0, x = \frac{2\pi}{3}, x = \frac{4\pi}{3}, x = 2\pi$), yields the approximation

$$y = 0.094266x^3 - 0.888436x^2 + 1.860735x$$

Using the GASOPE method, the result over 100 simulations (initialised using table 3.2 and a maximum polynomial order of 3) is consistently

$$y = 0.0943367x^3 - 0.889542x^2 + 1.93574x - 0.226775$$

The difference in MSE between these two methods is 0.01552, with the GASOPE method being the more accurate of the two. The GASOPE method decided, in this case, to include the constant -0.226775 because the gain in accuracy was significant.

Both of the above results illustrate that the GASOPE method does indeed find the optimal polynomial approximation to a function.

3.5 CONCLUSION

This chapter presented and discussed a genetic algorithm approach to evolve structurally optimal polynomial expressions (GASOPE) to represent a given data set. The genetic algorithm was shown to be significantly faster than a neural network approach, and the genetic algorithm produced comparable results when compared to the neural network approach in terms of generalisation ability for most of the functions used in this chapter on both clean and noisy datasets (which included chaotic time-series). The success of the genetic algorithm approach is mainly due to the specialised mutation and crossover operators, and can also be attributed to the fast k-means clustering algorithm, both of which lead to a significant reduction of the search space. Performance gains in terms of speed can also be attributed to the highly parallel search behaviour of genetic algorithms, *i.e.* the increase in performance of the GASOPE method over a neural network is not just a function of the number of patterns presented.

Although the genetic algorithm discussed in this chapter appears to be fairly effective both in terms of accuracy and speed, there is, however, one serious drawback. The drawback is that polynomial structures are poor predictors of periodic data. In order to predict periodic data over an interval with polynomials, it is necessary to increase the order of the interpolating polynomial. However, such a prediction becomes particularly poor and deteriorates when the polynomial predictor is used to extrapolate information outside of the aforementioned interval. This problem can be solved in one of 2 ways: Build an expression that utilises a periodic function such as cosine or sine, or use only linear predictors at the ends of the approximation interval.

The use of cosine or sine as a periodic function in the above, would require a substantial rework of the data structure employed to house term-coefficient pairs. The data structure would have to be changed from a list to a tree, which would require the operators to be changed. The use of linear predictors at the ends of the approximation interval is fairly simple to implement: Construct an expression that represents a hyper-plane in the attribute space, *e.g.* if two inputs (x_0, x_1) and one target output z are used, construct the expression $z = r_0 x_0 + r_1 x_1 + c$ and use the linear system mentioned in this chapter to solve r_0, r_1 and c. This hyper-plane can then be used in conjunction with an interval measure to extrapolate any unseen data outside the training interval.

The next chapter presents a genetic program for the mining of continuous-valued classes

(GPMCC) that utilises the expressions generated by the GASOPE method to provide multivariate models for the leaf nodes of a model tree.

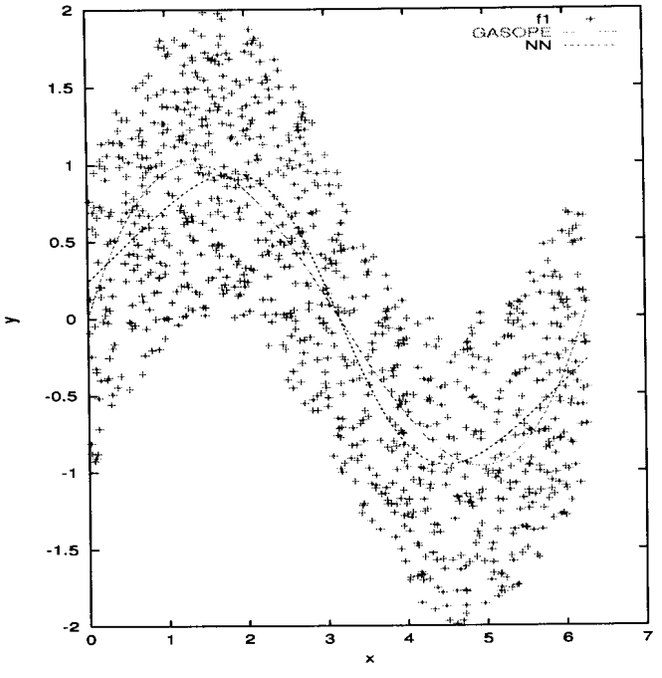

Figure 3.7: Function f1 actual *vs.* GASOPE and NN predicted

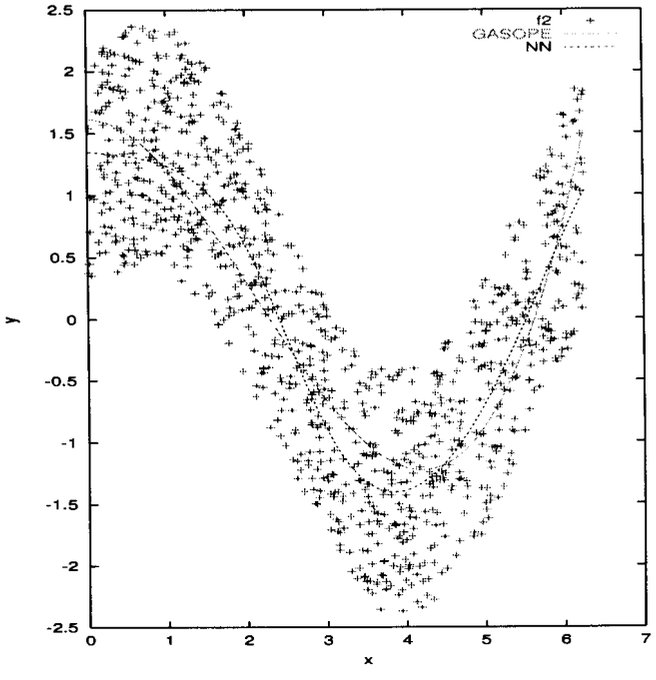

Figure 3.8: Function f2 actual *vs.* GASOPE and NN predicted

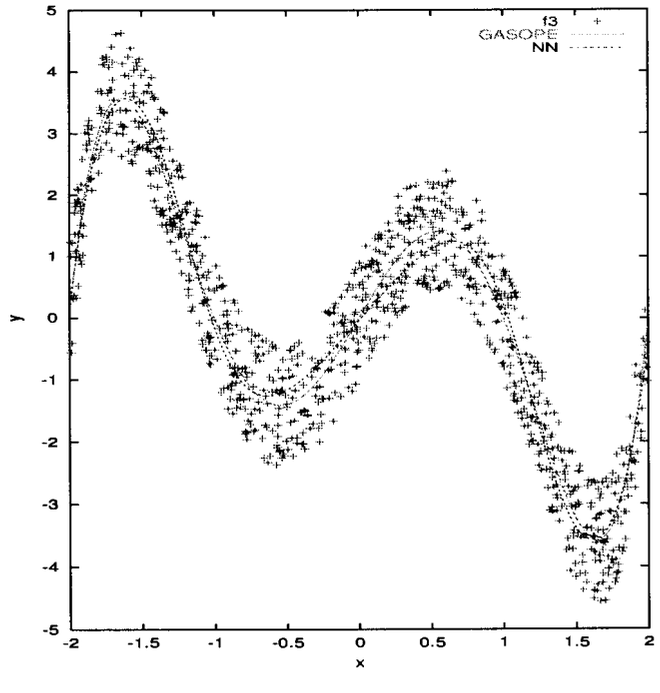

Figure 3.9: Function f3 actual *vs.* GASOPE and NN predicted

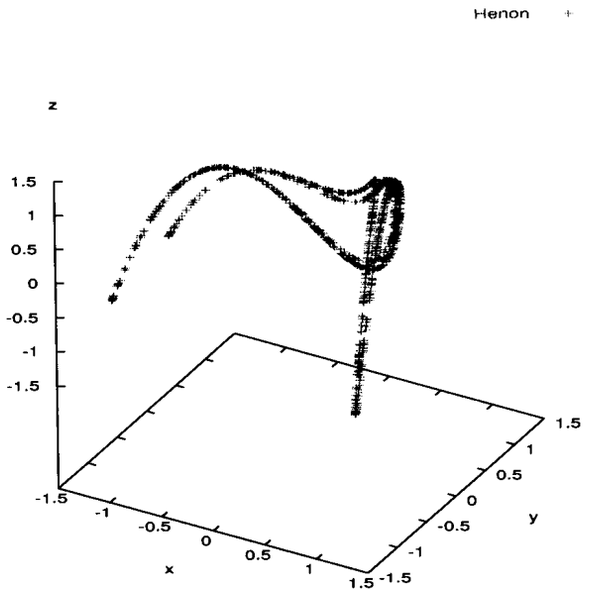

Figure 3.10: Henon map

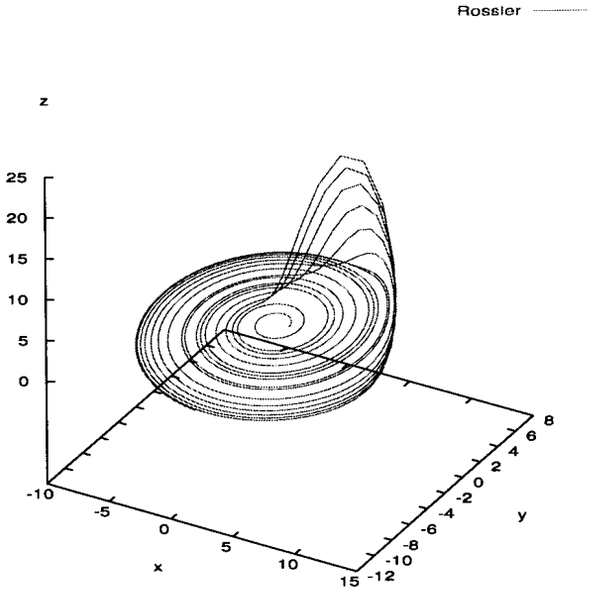

Figure 3.11: Rossler attractor

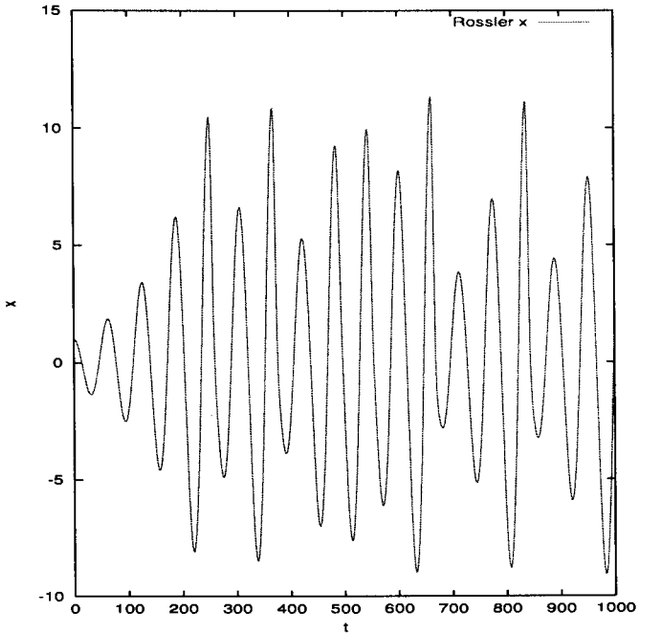

Figure 3.12: Rossler attractor: x component

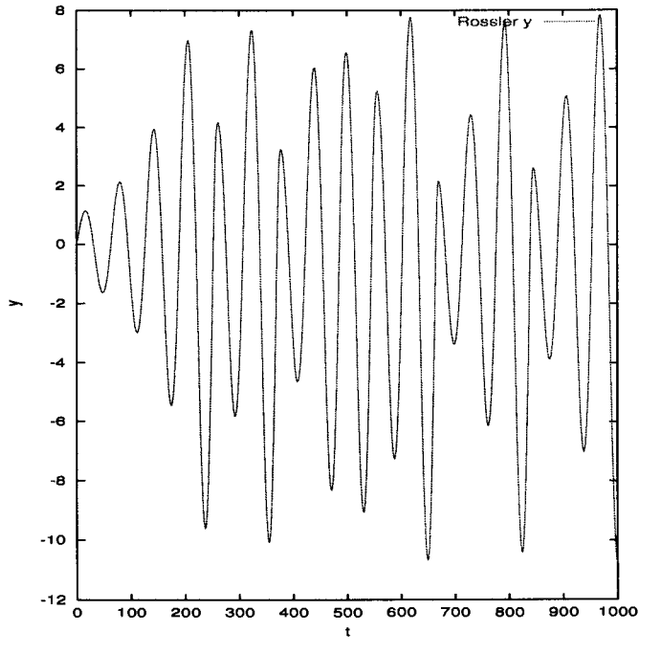

Figure 3.13: Rossler attractor: y component

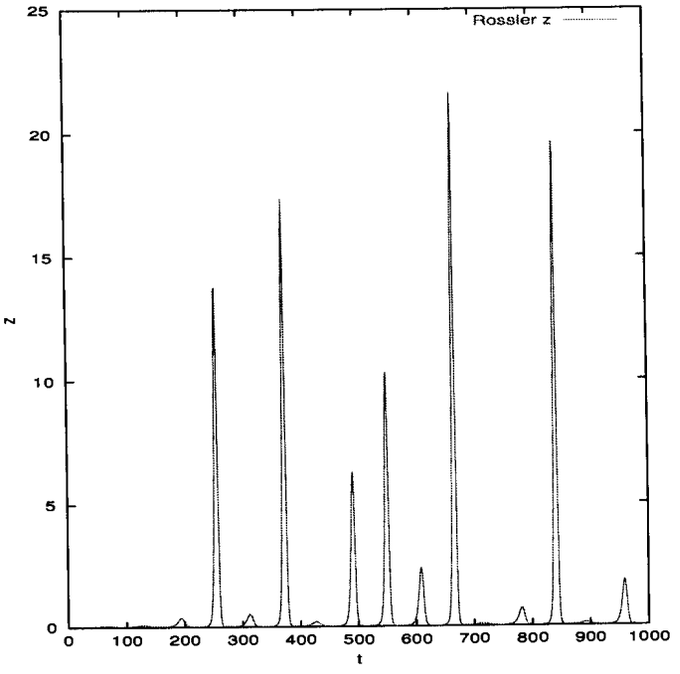

Figure 3.14: Rossler attractor: z component

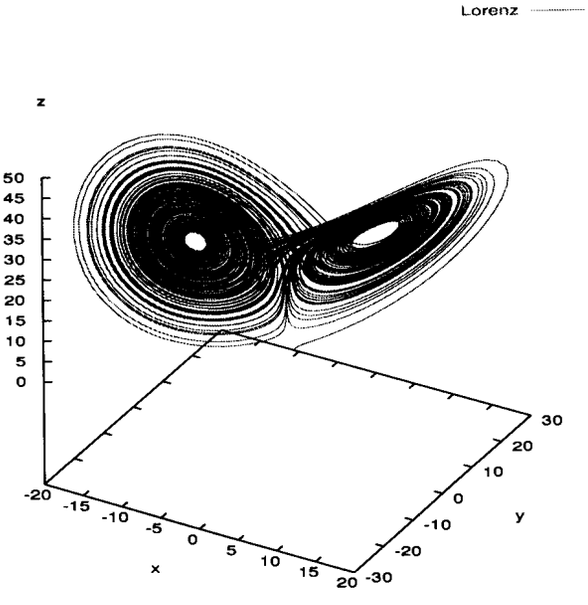

Figure 3.15: Lorenz attractor

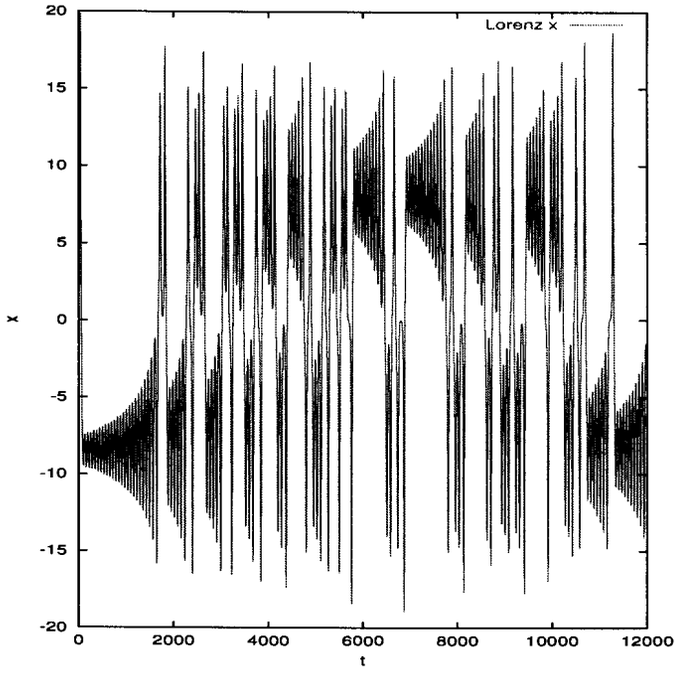

Figure 3.16: Lorenz attractor: x component

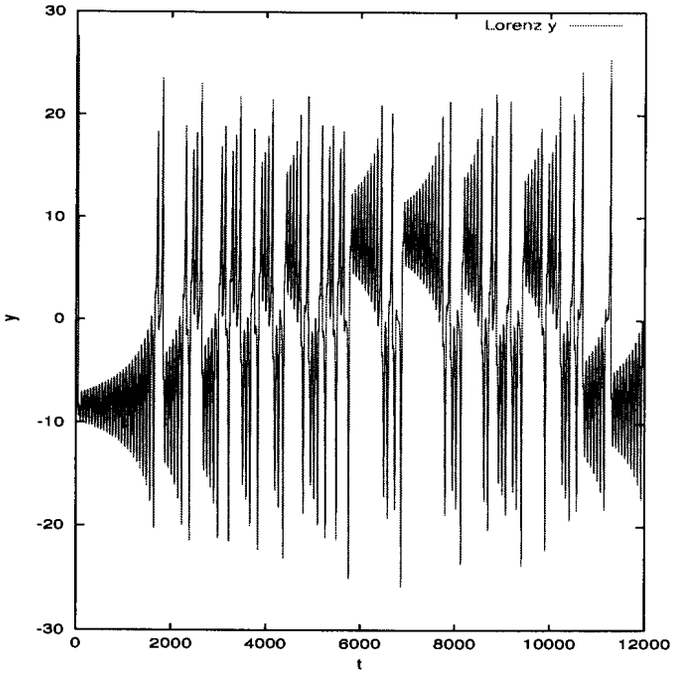

Figure 3.17: Lorenz attractor: y component

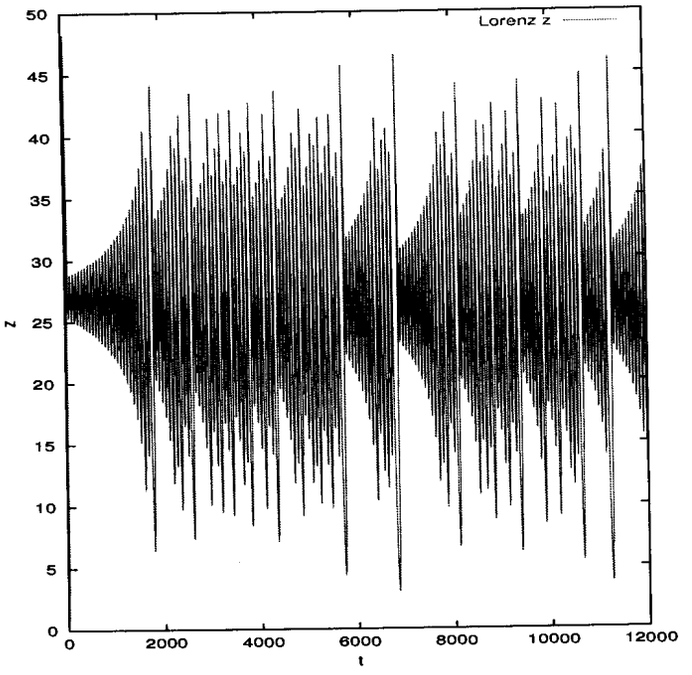

Figure 3.18: Lorenz attractor: z component

Chapter 4

THE GPMCC METHOD

The previous chapter presented a genetic algorithm for evolving structurally optimal polynomial expressions (GASOPE). This chapter discusses a genetic program for the mining of continuous-valued classes (GPMCC). The GPMCC method relies heavily on the algorithms presented earlier in this thesis.

4.1 INTRODUCTION

Knowledge discovery algorithms like C4.5 [83] and M5 [82] utilise metrics based on information theory to partition the problem domain and to generate rules. However, these algorithms implement a *greedy* search algorithm to partition the problem domain. For a given attribute space, C4.5 and M5 attempt to select a test that minimises the relative entropy of the subsets resulting from the split. This process is applied recursively until all subsets are homogeneous or some accuracy threshold is reached.

Figure 4.1 illustrates the partitioning problem by drawing parallels with graph theory. The minimal spanning tree of the graph of figure 4.1 illustrates how a greedy algorithm reaches point D from point A through point B, which is clearly not the optimal path. The optimal path from point A to point D traverses point C. This clearly shows that knowledge discovery algorithms such as C4.5 and M5 may not generate the smallest possible number of rules for a given problem. A large number of rules results in decreased comprehensibility, which violates one of the prime objectives of data mining.

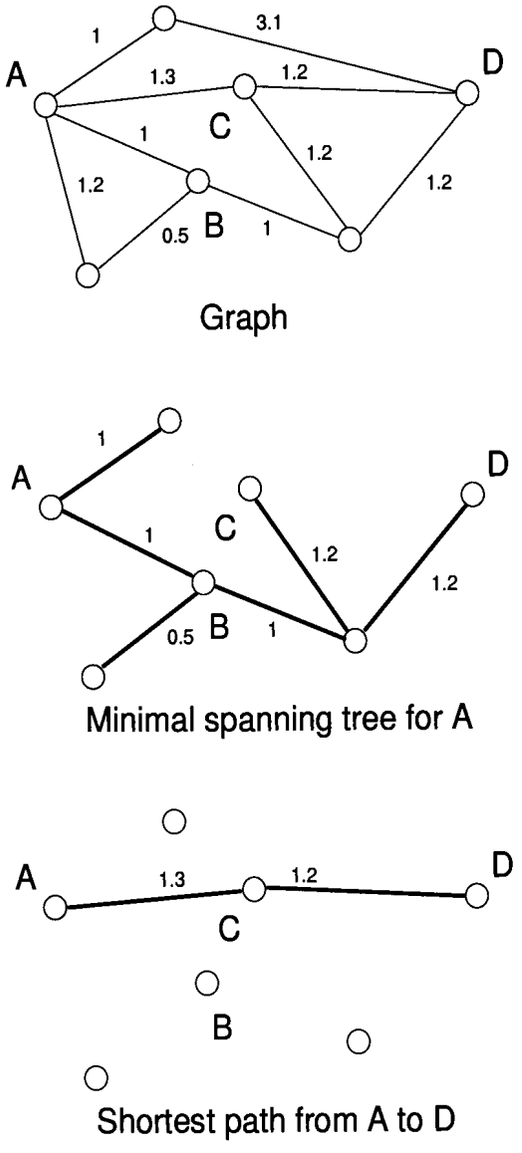

Figure 4.1: An arbitrary graph

This chapter discusses a regression technique that does not implement a greedy search algorithm. The regression technique utilises a genetic program for the mining of continuous valued classes (GPMCC) which is suitable for mining large databases. Although the majority of continuous data is linear, there are cases for which a non-linear approximation technique could be useful, *e.g.* time-series. Therefore, the GPMCC method utilises the GASOPE method

of chapter 3 to provide non-linear approximations (models) to be used as the leaf nodes (terminal symbols) of a model tree.

The remainder of this chapter is organised as follows: Section 4.2 provides a background to techniques that do not implement greedy search algorithms to generate rules. The structure and implementation specifics of the GPMCC method are discussed in section 4.3. Section 4.4 presents the experimental findings of the GPMCC method for a number of real-world and artificial databases. Finally, section 4.5 presents the summarised findings and envisioned future developments to the GPMCC method.

4.2 BACKGROUND

The previous section presented a brief introduction to the GPMCC method. This section presents a detailed discussion of two existing methods suitable for non-linear regression. A novel method of generating comprehensible regression rules from a trained artificial neural network is discussed in section 4.2.1. Finally, section 4.2.2 presents genetic programming approaches for non-linear regression and model tree induction.

4.2.1 Artificial neural networks

Artificial neural networks (ANNs) are widely used as a tool for solving regression problems. However, ANNs have one critical drawback: the complex input to output mapping of the ANN is almost impossible for a human user to comprehend. ANNs are thus one of a handful of *black-box* methods that do not satisfy the comprehensibility requirement of knowledge discovery. This section discusses a recent work by Setiono that allows decision rules to be generated for regression problems from a trained ANN, called NeuroLinear [91][94]. Setiono's findings indicated that rules extracted from ANNs were more accurate than those extracted by various discretisation methods.

This section is divided into three parts. *Network training and pruning* discusses the training and pruning strategy of the ANN used by Setiono. *Activation function approximation* describes how a piecewise linear approximation of the activation function is obtained. *Generation of rules* discusses the algorithm for generating rules from a trained ANN.

Network training and pruning

The method starts by training an ANN that utilises hyperbolic tangent activation functions in the hidden layer (of size H). Training is performed on a training set of $|P|$ training points $(\mathbf{x}_i, y_i), i = 1, \cdots, |P|$ where $\mathbf{x}_i \in \Re^N$ and $y_i \in \Re$. Training, in this case, minimises the sum of squares error $E_{SS}(\mathbf{w}, \mathbf{v})$ augmented with a penalty term $P(\mathbf{w}, \mathbf{v})$.

$$E_{SS}(\mathbf{w}, \mathbf{v}) = \sum_{i=1}^{|P|} (y_i - y_i^{\bullet})^2 + P(\mathbf{w}, \mathbf{v}) \tag{4.1}$$

with

$$\begin{aligned} P(\mathbf{w}, \mathbf{v}) \quad = \quad & \varepsilon_1 \left(\sum_{m=1}^{H} \left(\sum_{l=1}^{N} \frac{\eta w_{ml}^2}{1+\eta w_{ml}^2} + \frac{\eta v_m^2}{1+\eta v_m^2} \right) \right) + \\ & \varepsilon_2 \left(\sum_{m=1}^{H} \left(\sum_{l=1}^{N} w_{ml}^2 + v_m^2 \right) \right) \end{aligned} \tag{4.2}$$

where $\varepsilon_1, \varepsilon_2$ and η are positive penalty terms, $y_i^{\bullet}$ is the predicted output for input sample $\mathbf{x}_i$, *i.e.*

$$y_i^{\bullet} = \sum_{m=1}^{H} \tanh((\mathbf{x}_i)^T v_m) + \tau, \tag{4.3}$$

$\mathbf{w}_m \in \Re^N$ is the vector of network weights from the input units to hidden unit m, w_{ml} is the l-th component of $\mathbf{w}_m$, $v_m \in \Re$ is the network weight from the hidden unit m to the output unit and τ is the output unit's bias.

Setiono performed training using the BFGS optimisation algorithm, due to its faster convergence than gradient descent [29][92]. After training, irrelevant and redundant neurons were removed from the ANN using the N2PFA (Neural Network Pruning for Function Approximation) algorithm [93]. ANN pruning prevents the ANN from over-fitting the training data (discussed in section 2.3.3) and also reduces the length of the rules extracted from the ANN. The length of the extracted rules are reduced because the number of variables (weights, input units and hidden units) affecting the outcome of the ANN are reduced.

Activation function approximation

The complex input to output mapping of an ANN is a direct consequence of using either the hyperbolic tangent or the sigmoid function as artificial neuron activation functions. However, as was discussed in section 2.3.2, the importance of these functions in ANN training is that they are monotonically increasing, differentiable and continuous throughout their domains.

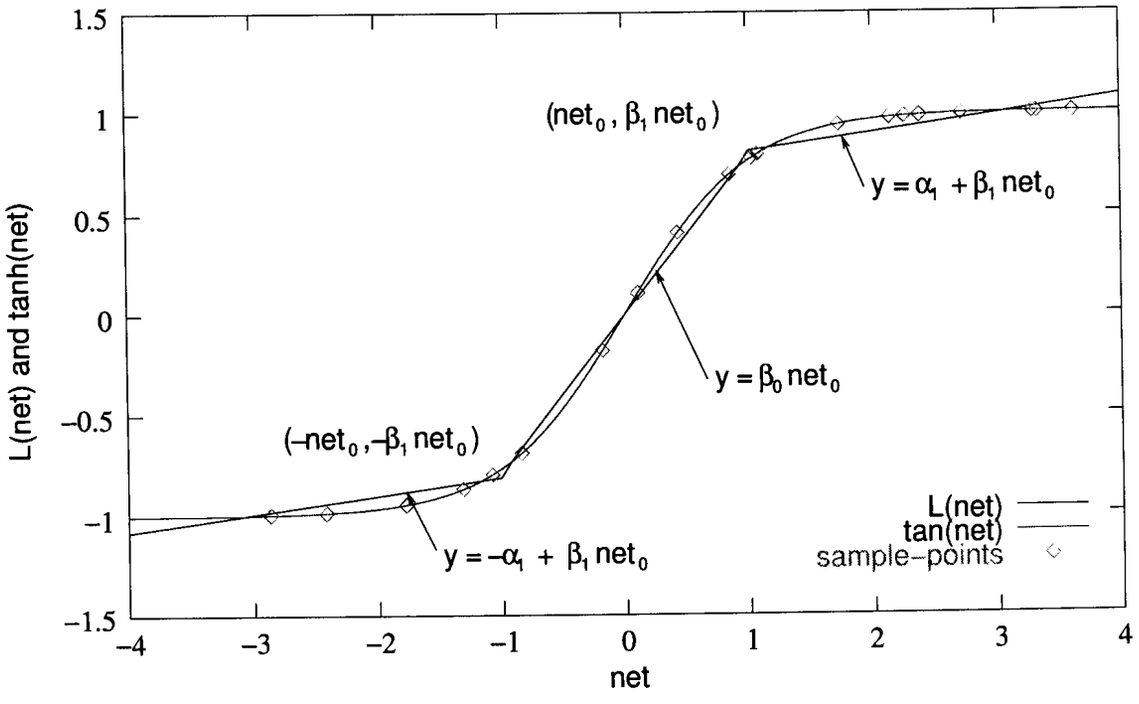

Figure 4.2: A 3-piece linear approximation of the hidden unit activation function $\tanh(net)$ given 20 training samples ($\diamond$)

In order to generate comprehensible rules, a 3-piece linear approximation of the hyperbolic tangent activation function is constructed. This entails finding the cut-off points (net_0 and $-net_0$), the slope (β_0 and β_1) and the intersection (0, α_1 and $-\alpha_1$) of each of the three line segments. The sum squared error between the 3-piece linear approximation and the activation function is minimised to obtain the values for each of these parameters:

$$\min_{net_0,\beta_0,\beta_1,\alpha_1} = \sum_{i=1}^{|P|} (\tanh(net_i) - L(net_i))^2$$

where $net_i = \mathbf{x}_i^T \cdot \mathbf{w}$ is the weighted input of sample i and

$$L(net) = \begin{cases} -\alpha_1 + \beta_1 net & \text{if} \quad net < -net_0 \\ \beta_0 net & \text{if} \quad -net_0 \leq net \leq net_0 \\ \alpha_1 + \beta_1 net & \text{if} \quad net > net_0 \end{cases}$$

The intercept and slopes which minimises the sum squared error are calculated as follows:

$$\beta_0 = \frac{\sum_{|net_i| \leq net_0} net_i \tanh(net_i)}{\sum_{|net_i| \leq net_0} net_i^2}$$

$$\beta_1 = \frac{\sum_{|net_i| > net_0} (net_i - net_0)(\tanh(net_i) - \tanh(net_0))}{\sum_{|net_i| > net_0} (net_i - net_0)^2}$$

$$\alpha_1 = (\beta_0 - \beta_1) net_0$$

The weighted input net_i of each sample is checked as a possible optimal value for net_0 starting from the one that has the smallest magnitude. Figure 4.2 illustrates how the 3-piece linear approximation is constructed.

Generation of regression rules

Linear regression rules are generated from a pruned ANN once the network hidden unit activation functions $\tanh(net)$ has been approximated by the 3-piece linear function described above. The regression rules are generated as follows:

1. For each hidden unit $m = 1, \cdots, H$
   (a) Generate a 3-piece linear approximation $L_m(net)$.
   (b) Using the points $-net_{m0}$ and net_{m0} from function $L_m(net)$, divide the input space into 3 subregions.
2. For each non-empty subregion $r = 1, \cdots 3^H$, generate a rule as follows:
   (a) Define a linear equation that approximates the ANN's output for an input pattern i in subregion r as the rule *consequent*:

$$y_i^{\bullet} = \sum_{m=1}^{H} v_m L_m(net_{mi}) + \tau$$

   where $net_{mi} = \mathbf{x}_i^T \mathbf{w}_m$, $v_m \in \Re$ is the network weight from the hidden unit m to the output unit and τ is the output unit's bias.

(b) Generate the rule *antecedent*: $((C_1) \wedge \cdots \wedge (C_m) \wedge \cdots \wedge (C_H))$ where C_m is either $net_{mi} < -net_{m0}$, $net_{mi} > net_{m0}$ or $-net_{m0} < net_{mi} < net_{m0}$. Each C_m represents an attribute test. The antecedent is formed by the conjunction of the appropriate tests from each of the hidden units.

(c) Simplify the rule antecedent.

The rule antecedent $((C_1) \wedge \cdots \wedge (C_m) \wedge \cdots \wedge (C_H))$ defines the intersection of each subspace in the input space. For a large number of hidden ANs, this antecedent becomes large. If this antecedent is simplified, using logic or by using an algorithm such as C4.5 (discussed in section 2.1.2), then the rules generated by the above algorithm will be much easier to comprehend.

4.2.2 Genetic programming

The evolutionary computing paradigm of genetic programming (GP) can be used to solve a wide variety of data mining problems. This section discusses GP methods for symbolic regression, decision, regression and model tree induction, and scaling problems associated with GP. GP satisfies the comprehensibility requirement of knowledge discovery, because the representation of an individual (or chromosome) can be engineered to provide easily understandable results.

Symbolic regression

Unlike artificial neural networks, GP can be used to perform *symbolic regression* without the need for data transformations. GP is also capable of regression analysis on variables that exhibit non-linear relationships, as apposed to the linear regression techniques presented in section 3.2. GP is thus a useful tool for regression analysis of non-linear data. However, because most data is in fact linear, a more conventional form of regression should always be considered first.

Regression problems are solved by fitting a function, represented by a chromosome, to the dataset using a fitness function that minimises the error between them. The terminal set is defined as a number of constants and attributes, *e.g.* $\{32, 2.5, 0.833, x, y, z\}$, and describes a number of valid states for the leaf nodes of a chromosome (in the form of a tree). The function set is defined by a number of operators, *e.g.* $\{+, -, *, \backslash, cos, sin\}$, and describes a number of

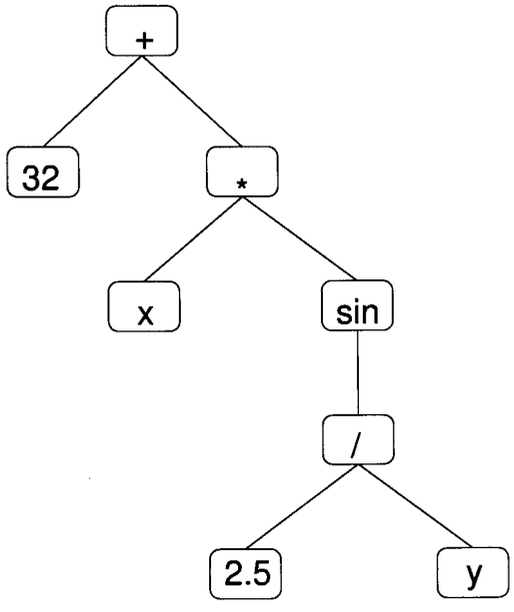

Figure 4.3: An example chromosome for a regression problem: terminal set $\{32, 2.5, 0.833, x, y, z\}$ and function set $\{+, -, *, \backslash, cos, sin\}$

valid states for the internal nodes of a chromosome. Figure 4.3 demonstrates an example chromosome for the aforementioned terminal set and function set.

The constants used in the terminal set are an Achilles' Heel of a symbolic regression genetic program. If a population is liberally scattered with constants chosen from a preset range, *e.g.* $[-1, 1]$, it may be difficult for a genetic program to evolve the expression $300x$. Abass *et al.* present a concise overview of methods to correct this problem [1].

Decision, regression and model tree induction

A number of different classification systems that utilise genetic programming (GP) have been developed. This section discusses two interesting ones. Additionally, this section shows how a genetic program can be developed to directly evolve decision, regression and model trees.

Eggermont *et al.* present a GP approach that utilises a *stepwise adaptation of weights* (SAW) technique in order to learn the optimal penalisation factor for the GP fitness function [30]. Each individual in the population represents a classification rule, and utilises a function set of boolean connectives and a terminal set of attribute tests, *i.e.* either the rule condition

covers a training pattern, in which case it is asserted to belong to a class, or it does not. The approach was shown to have increased accuracy over the fixed penalisation factor case.

Freitas presents an interesting GP framework that evolves SQL queries in order to increase scalability, security and portability [41]. Each individual consists of two parts: a tuple-set descriptor and a goal attribute. The GP approach utilises a niching strategy in order to force the method to produce innovative rules.

GP can also be used to directly build decision, regression and model trees. As was mentioned in section 4.2.1, artificial neural networks provide no comprehensible explanation of how they classify or approximate a dataset. On the other hand, classification systems, such as C4.5 (from section 2.1.2), and regression systems, such as M5 (from section 2.1.2), generate overly complex trees. GP is potentially capable of providing a compromise between these two extremes.

For zero-order learning, the function set of the chromosome consists of a number of attribute tests. The terminal set for the chromosome consists of either

- a number of **classes** for decision trees,
- a number **values** for regression trees,
- or a number of **models** for model trees.

The models for model trees can be obtained by linear regression, symbolic regression or the GASOPE method of the previous chapter.

The fitness function can either

- evaluate the number of cases that are correctly classified for a decision tree,
- or minimise the error between the predicted response of the individual and the target response of a number of training patterns.

Additionally, the fitness function should implement a penalisation factor in order to penalise the complexity of a chromosome. In this manner, genetic programs can be used to minimise both the bias and the variance of the model described by a chromosome.

Scaling problems

GP has shown considerable promise in its problem solving ability over a wide range of applications including data mining [45][62]. However, problems exists in scaling GP to larger problems such as data mining. Marmelstein and Lamont summarise many of these difficulties [73]. Some of the most important scaling problems are:

- GP performance is very dependent on the composition of the function and terminal sets.
- There is a trade-off between GP's ability to produce good solutions and parsimony.
- The size and complexity of GP solutions can make it difficult to understand. Furthermore, solutions can become bloated with extraneous code (also known as introns).

Of the above difficulties, the most difficult to control is the complexity of GP solutions, otherwise known as code growth. Abass describes many methods for the removal of introns, *e.g.* chromosome parsing, alternative selection methods and alternative crossover methods [1].

4.3 GPMCC STRUCTURE

This section discusses a genetic program for the mining of continuous-valued classes (GPMCC). Section 4.3.1 presents an overview of the GPMCC method and its various components. An iterative learning strategy used by the GPMCC method is discussed in section 4.3.2. Section 4.3.3 describes, in detail, the fragment pool utilised by the GPMCC method. Finally, the core genetic program for model tree induction is discussed in section 4.3.4.

4.3.1 Overview

The genetic program for the mining of continuous-valued classes (GPMCC) consists of three parts:

1. An iterative learning strategy to reduce the number of patterns that are presented to the genetic program.

2. A pool of GASOPE fragments, which serve as a terminal set for the terminal nodes of a chromosome in a genetic program. This pool of fragments is evolved using mutation and crossover operators.

3. A genetic program to evolve model trees.

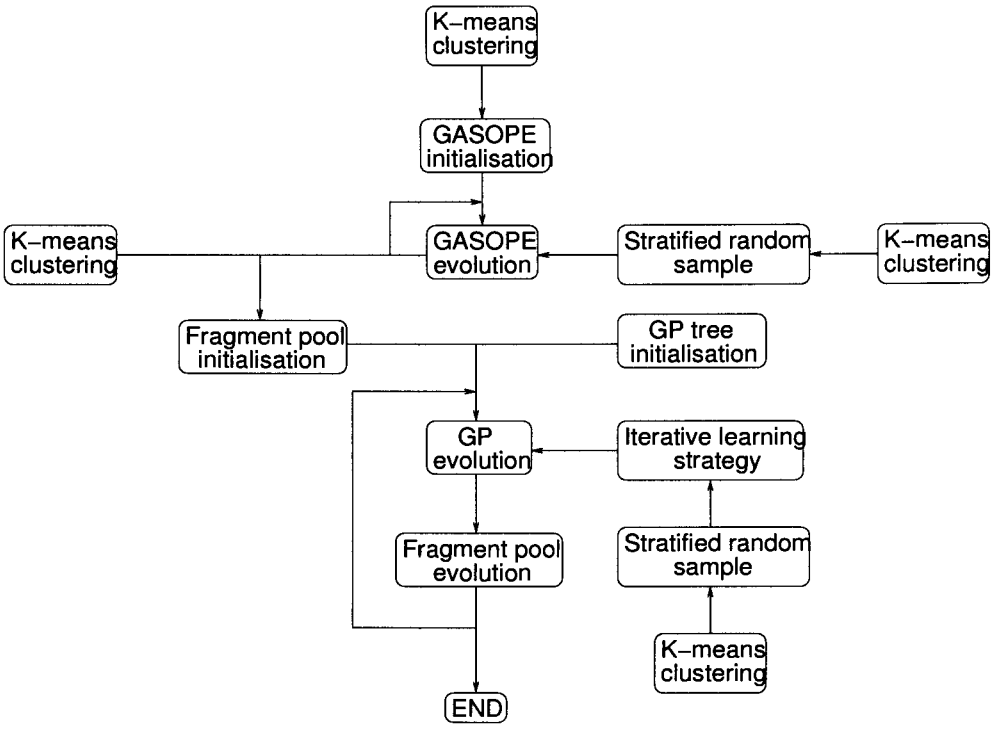

Figure 4.4: Overview of the GPMCC learning process

Figure 4.3.1 shows an overview of the GPMCC learning process. In addition, the GPMCC learning process is summarised below:

1. Let $g = 0$ be the generation counter

2. Initialise a population $G_{GP,g}$ of N individuals, *i.e.*

$$G_g = \{I_\chi | \chi = 1, \cdots, N\}$$

3. While $g < M$, where M is the total number of generations, do

(a) Sample S, using an incremental training strategy, from the remaining training patterns P, *i.e.* $P' \subset P, S = S \cup P'$.

(b) Remove the sampled patterns from P, *i.e.* $P = P/S$.

(c) Evaluate the fitness $R_a^2(I_\chi)$ of each individual in population $G_{GP,g}$ using the patterns in S

(d) Let $G'_{GP,g} \subset G_{GP,g}$ be the top $x\%$ of the individuals, based on fitness, to be involved in elitism

(e) Install the members of $G'_{GP,g}$ into $G_{GP,g+1}$

(f) Let $G''_{GP,g} \subset G_{GP,g}$ be the top $n\%$ of the individuals, based on fitness, to be involved in crossover

(g) Run the fragment pool optimisation algorithm once

(h) Perform crossover:

i. Randomly select two individuals I_α and I_β from $G''_{GP,g}$

ii. Produce offspring I_γ from I_α and I_β

iii. Install I_γ into $G'''_{GP,g}$

(i) Perform mutation:

i. Select an individual I_ω from $G'''_{GP,g}$

ii. Mutate I_ω by randomly selecting a mutation operator to perform.

iii. Install I_ω into $G_{GP,g+1}$

(j) Evolve the next generation $g := g + 1$

4.3.2 Iterative learning strategy

The GPMCC method utilises an iterative learning strategy to reduce the number of training patterns presentations per generation. Additionally, the iterative learning strategy should result in more accurate rules being generated [32][83]. The strategy utilises the k-means clustering of section 3.3.1 to cluster the training data. Clustering finds regions of high pattern density. As was discussed in section 3.3.1, larger clusters form around the turning points of a function. Using a proportional sampling strategy, the most informative patterns can be selected for training.

As with the GASOPE method, a *stratified random sample* is selected from the available training patterns during each generation of the genetic program. The size, s, of the initial sample is chosen as a percentage of the total number of training patterns $|P|$. The sampling strategy utilises an acceleration rate to increase the size of the sample during every generation of the genetic program. This size of the sample is increased until the size of the sample equals the total number of training patterns.

The stratified random sample is drawn proportionally from each of the k clusters of the k-means clusterer, *i.e.*:

$$S = S \cup_{\delta=1}^{k} C'_{\delta}$$

where

$$C'_{\delta} \subset C_{\delta} : |C'_{\delta}| = \frac{|C_{\delta}| \cdot acceleration \cdot s}{|P|}$$

and $|C_{\delta}|$ is the (stratum) size of cluster C_{δ}, $|P|$ is the size of the data set, s is the initial sample size and *acceleration* is the acceleration rate. The acceleration rate increases the size of the sample at each generation.

4.3.3 The fragment pool

From section 2.1.2, a model tree is a decision tree that implements multi-variate linear models at its terminal nodes. However, linear models may not adequately describe time-series data or non-linear data. In these cases, a model tree that implements multi-variate linear models at its terminal nodes will perform a piecewise approximation of the problem space. Although such a piecewise approximation may be accurate, the number of rules induced by the approximation will be large as indicated by the results in section 4.4.3.

A number of techniques exist to perform non-linear regression. Two obvious non-linear regression techniques include

- using a genetic program to perform a symbolic regression (from section 4.2.2) of the data covered by the terminal nodes,
- or using the GASOPE method (from section 3.3.2) to perform a non-linear regression of the data covered by the terminal nodes.

However, the use of both of these methods can be shown to have a severe impact on the time taken to construct a model tree.

From section 3.4.3, the largest training time of the GASOPE method was approximately 1.5 seconds. Assuming a genetic program is used to construct a model tree, if a model is generated by a mutation operator, a 1.5 second time penalty will be incurred every time that mutation operator is called. If, for example, there is a 1% chance of the mutation operator being run on an individual in a population of 100 individuals with, on average, 5 terminal nodes per individual, a 7.5 second time penalty will be incurred per generation ($0.01 \times 100 \times 5 \times 1.5 = 7.5$). For 1000 generations this performance penalty is 7500 seconds (2 hours and 5 minutes). In other words, a very large proportion of the genetic program's training time will be spent optimising the models.

This section discusses the *fragment pool*. The fragment pool is an evolutionary algorithm for improving the time taken for a model to be generated, based on context modelling. The fragment pool represents a *belief space* of terminal symbols for a model tree. The remainder of this section discusses the *representation* of the fragment pool, the *initialisation* of a fragment, the fragment *mutation* and *crossover* operators, and the *fitness function*.

The implementations of the fragment pool and the genetic program of section 4.3.4 to evolve model trees are heavily intertwined. Therefore, for the remainder of this section assume that there is a genetic program that evolves model trees, whose terminal nodes are models that are obtained from the fragment pool.

Representation

Each *fragment* in the pool represents a model-lifetime mapping

$$F_\omega = \{I_\omega \rightarrow \pi_\omega\}$$

where I_ω is a GASOPE individual from section 3.3.2 (model) and π_ω is the lifetime of I_ω. The lifetime π_ω represents the age of a fragment in the fragment pool. When a fragment's lifetime expires, it is removed from the pool. The lifetime of a fragment can, however, be reset if the fragment is deemed "useful". By counting the number of times a model appears as a terminal node in the members of the crossover group of the genetic program, the fragment usefulness can be determined. A model is more likely to appear as a terminal node of members of the

crossover group if the model closely approximates the sub-space described by the training and validation patterns covered by the path to the terminal node. Thus, the fragment pool implements a kind of *context modelling* [88], because fragments that result in sub-optimal approximations for these sub-spaces are eventually removed and can no longer contaminate the pool.

Initialisation

The initialisation of the fragment pool G_{FP} proceeds as follows:

1. Let $k = initial\_clusters$
2. Perform k-means clustering (from section 3.3.1) to obtain k clusters $C_\delta, \delta = 1, \cdots, k$.
3. For each cluster $C_\delta, \delta = 1, \cdots, k$
   (a) Use the GASOPE method (from section 3.3.2) to obtain a non-linear regression I_δ of the patterns in each cluster.
   (b) Insert the fragment $F_\delta = \{I_\delta \rightarrow 0\}$ into the fragment pool $G_{FP} = G_{FP} \cup F_\delta$.
4. Divide k by the split factor *i.e* $k := \frac{k}{split\_factor}$.
5. Return to step 2 while $k > 1$.

Essentially, the above algorithm performs multiple piecewise approximations of the problem space to build an initial set of fragments. The number of initial clusters, $initial\_clusters$, and the split factor, $split\_factor$, ultimately control the total number of piecewise approximations (models) that are generated. The algorithm starts by fitting many highly specific approximations and then increasing the approximation generality by decreasing the available number of clusters k. Decreasing the available number of clusters results in an increase in the number of patterns covered by each cluster centroid. This, in turn, results in a more general function approximation per cluster.

The models contained within the fragments form part of a terminal set for the genetic program. Whenever the genetic program requires a model, the fragment pool randomly selects a fragment and passes the fragment's model to the genetic program.

Mutation and crossover operators

The fragment pool mutation operators serve to inject additional fragments into the fragment pool. The addition of fragments to the fragment pool result in an increased number of models in the terminal set of the genetic program. Additional models in the terminal set prevents the stagnation of the genetic program, by allowing the introduction of models that approximate regions not covered by the initialisation of the fragment pool. Additionally, the mutation operators also serve to fine-tune the models at the terminal nodes of the genetic program. Two mutation operators exist for the fragment pool;

- the **shrink** operator
- and the **introduce** operator.

The shrink operator duplicates an arbitrary fragment in the fragment pool, and applies the shrink operator of section 3.3.2 to the duplicate. The introduce operator calls the GASOPE optimisation algorithm of section 3.3.2 on the training and validation patterns covered by the *path* of a terminal node of an arbitrary individual of the genetic program. The path is defined as the nodes traversed from the root of a tree to a specific node in the tree. The model obtained from the GASOPE optimisation algorithm is given a lifetime of 0 and is inserted into the fragment pool. The other mutation operators of the GASOPE method are not used by the fragment pool optimisation process, because they require the coefficients for a term to be calculated. The coefficients can only be calculated if the training and validation sets are kept for each fragment. This, however, is not feasible because a fragment in the pool may be utilised by more than one individual in the genetic program, *i.e.* an individual could describe multiple sub-spaces of the problem domain. All the operators of the GASOPE method are, however, used in the initialisation of the fragment pool.

The crossover operator of the fragment pool is an invocation of the GASOPE crossover operator of section 3.3.2. A fragment with a non-zero usefulness factor and an arbitrary fragment are randomly chosen from the fragment pool and their models are given to the GASOPE crossover operator. The model obtained from the GASOPE crossover operator is given a lifetime of 0 and is inserted into the fragment pool.

A culling operator removes fragments from the fragment pool, when those fragment's fragment lifetimes have expired, *i.e.* when the fragment lifetimes are larger than some upper-

bound. The culling operator removes fragments from the fragment pool that have not been useful for a number of generations. This operator ensures that the fragment pool does become uncontrollably large. A large fragment pool results in a large number of terminal symbols, which may increase the time taken to optimise the genetic program.

The shrink operator and the crossover operator are uniformly/randomly applied once after the completion of a generation of the genetic program. This ensures that the size of the fragment pool increases at least once per generation, in order to counteract the effects of the fragment lifetime, *i.e.* the removal of useless fragments. The introduce operator is applied with a statistical probability whenever the genetic program requires a model, *i.e.* when the relevant genetic program mutation operator is invoked. If the fragment lifetime is too large or the introduce operator is applied too often, the fragment pool will grow too quickly. A large fragment pool results in a large number of terminal symbols, which may have a negative consequence on the convergence properties of the genetic program. Conversely, a small fragment pool may lead to the stagnation of the genetic program, because there may not be enough variation in the terminal symbols described by the fragment pool.

Fitness function

The fitness function rewards the usefulness of a fragment. As was mentioned earlier, the fragment usefulness is determined by counting the number of times a fragment appears as a terminal node of individuals in the crossover group of the genetic program. The crossover group consists of the top individuals in the genetic program, obtained through tournament selection. Thus, the usefulness of a fragment is determined by the number of times it appears as well as where it appears, *i.e.* fragments not used as terminal symbols of the crossover group are useless, because the overall fitness of the individuals using those fragments is poor.

Fragment pool optimisation algorithm

The optimisation algorithm for the fragment pool is as follows:

1. Obtain a set $G''_{GP,g}$ of individuals from the crossover group of a genetic program.
2. Evaluate the fitness of each fragment F_ω in the pool G_{FP} using $G''_{GP,g}$, *i.e.* for each fragment $F_\omega \in G_{FP}$:

(a) Count the number of times, n, the individual I_ω appears as a terminal symbol in $G''_{GP,g}$.

(b) Set the fitness of the individual $F_{FP}(F_\omega) = n$.

3. Select a crossover group from the pool $G'_{FP} \subset G_{FP}$, where $F_{FP}(F_\omega) > 0, F_\omega \in G'_{FP}$ (the fragments of G'_{FP} still reside in G_{FP}).

4. Reset the fragment lifetime of all fragments in GP'_{FP} to 0, as they were deemed useful.

5. Increase the fragment lifetime of all fragments in GP_{FP}/GP'_{FP} by one.

6. Remove/cull any fragment from G_{FP} whose fragment lifetime has expired.

7. If $\mathrm{U}(0,1) < 0.5$

   (a) Select a fragment F_α from G'_{FP} and a fragment F_β from G_{FP}.

   (b) Perform crossover to obtain a fragment $F_\gamma = \{I_\gamma \rightarrow 0\}$.

   (c) Insert F_γ into G_{FP}

8. Otherwise,

   (a) Select a fragment F_ω from G_{FP}.

   (b) Duplicate F_ω to get F'_ω.

   (c) Perform mutation on F'_ω.

   (d) Insert F'_ω into G_{FP}

The fragment pool optimisation algorithm is invoked at each generation by the genetic program.

4.3.4 Genetic program for model tree induction

This section discusses, in detail, a genetic program for inducing model trees. The models for this genetic program are obtained from the fragment pool discussed in section 4.3.3.

Representation

Each individual I_χ in the population represents a model tree such that:

$$I_\chi = NODE$$

where

$$NODE: \ (CONSEQUENT)|((ANTECEDENT \rightarrow NODE) \vee (\neg ANTECEDENT \rightarrow NODE))$$

$$ANTECEDENT: \ (NOMINAL\_ANTECEDENT)|(CONTINUOUS\_ANTECEDENT)$$

$$NOMINAL\_ANTECEDENT: \ (A_\xi = v_\xi)$$

$$CONTINUOUS\_ANTECEDENT: \ (A_\xi < v_\xi)|(A_\xi > v_\xi)|(A_\xi = v_\xi)|(A_\xi \neq v_\xi)$$

$$CONSEQUENT: \ (I_\omega)$$

A consequent, I_ω, represents a GASOPE model from the fragment pool, A_ξ represents a nominal-, continuous- or discrete-valued attribute and v_ξ represents a possible value of A_ξ. For the continuous antecedents, operators such as $\leq$ and $\geq$ are obtained by adjusting the attribute value v_ξ.

Initialisation

The GPMCC initialise operator creates an individual I_χ by recursively adding nodes to that individual, up to a maximum depth bound. The pseudo-code algorithm for the initialisation is as follows, where *CALLER* is a calling node initially set to *Nil* and *depth* is the maximum required depth of the tree:

Initialise: with parameters *CALLER* and *depth*

1. If $depth < maximum\_depth$ and $\mathrm{U}(0,1) < 0.5$
   (a) Select an attribute $A_\xi, \xi = 1, \cdots, I$ from the attribute space of dimension I.
   (b) If A_ξ is a continuous-valued attribute, select an operator $\mathrm{op}(\xi) \in \{<, >, =, \neq\}$.
   (c) Otherwise, $\mathrm{op}(\xi) \in \{=\}$.

(d) Select an attribute value $a_{\xi,i}$ for attribute A_ξ from a training pattern i, such that $v_\xi = a_{\xi,i}, i \in \{1, \cdots, |P|\}$.

(e) Create a node N_χ with antecedent $ant_\chi = (A_\xi \text{ op}(\xi) \text{ v}_\xi)$ and consequent $con_\chi = Nil$.

(f) Call **Initialise** with the node covered by the antecedent of N_χ (the left node), N_{ant_χ}, and depth $depth + 1$.

(g) Call **Initialise** with the node covered by the negation of the antecedent of N_χ (the right node), $N_{\neg ant_\chi}$, and depth $depth + 1$.

2. Otherwise,

(a) Select an individual I_ω from the fragment pool.

(b) Create a node N_χ with antecedent $ant_\chi = Nil$ and consequent $con_\chi = I_\omega$.

(c) Set the node covered by the antecedent of N_χ:

$$N_{ant_\chi} := Nil$$

(d) Set the node covered by the negation of the antecedent of N_χ:

$$N_{\neg ant_\chi} := Nil$$

3. Let $CALLER := N_\chi$ and return.

Once the procedure terminates, *CALLER* returns with the head of the tree. Figure 4.5 illustrates one outcome of the initialisation of an individual I_χ. Each node in the diagram is recursively initialised as N_χ and the arrows show the path taken by the initialisation method.

Mutation operators

The mutation operators serve to inject new genetic material into the population. Additionally, the mutation operators utilise domain specific knowledge (for reasons described in section 2.2.4) in order to improve the quality of the individuals in the population. A large number of mutation operators exist for the GPMCC method.

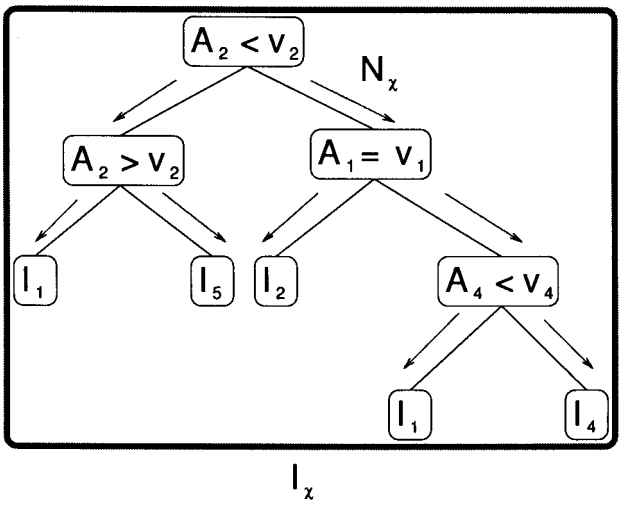

Figure 4.5: Illustration of GPMCC chromosome for an individual I_χ

- **Expand-worst-terminal-node operator**: The expand-worst-terminal-node operator locates and partitions the sub-space for which a terminal node has a higher relative error than all other terminal nodes in the individual I_χ. The relative error of the terminal node is determined by using the mean squared error E_{MS} between the model described by the terminal node and the training set covered by the path of that terminal node. The operator attempts to maximise the adjusted coefficient of determination R_a^2 (fitness) of the individual, by partitioning the sub-space described by a terminal node into smaller sub-spaces. The pseudo-code for the expand-worst-terminal-node operator is as follows:

  1. Select N_χ (shown in figure 4.6) such that $\forall N_i \in I_\chi : (con_\chi \neq Nil) \wedge (\mathrm{E_{MS}(N_\chi)} \leq \mathrm{E_{MS}(N_i)})$, *i.e.* select the worst terminal node.
  2. Select an attribute $A_\xi, \xi = 1, \cdots, I$ from the attribute space of size I (in order to turn the consequent into an antecedent).
  3. If A_ξ is a continuous-valued attribute, select an operator $\mathrm{op}(\xi) \in \{<, >, =, \neq\}$.
  4. Otherwise, $\mathrm{op}(\xi) \in \{=\}$.
  5. Select an attribute value $a_{\xi,i}$ for attribute A_ξ from a training pattern i, such that $v_\xi = a_{\xi,i}, i \in \{1, \cdots, |P|\}$.
  6. Set the antecedent ant_χ of node N_χ to $(A_\xi \ \mathrm{op}(\xi) \ \mathrm{v}_\xi)$.

7. Set the consequent con_χ of node N_χ to Nil (to satisfy the termination criteria).
8. Create a node covered by the antecedent of N_χ (the left node), N_{ant_χ}, with antecedent $ant_{\neg ant_\chi} = Nil$ and consequent $con_{ant_\chi} = I_\omega$.
9. Set the node covered by the antecedent of N_{ant_χ}:

$$N_{ant_{ant_\chi}} := Nil$$

10. Set the node covered by the negation of the antecedent of N_{ant_χ}:

$$N_{ant_{\neg ant_\chi}} := Nil$$

11. Create a node covered by the negation of the antecedent of N_χ (the right node), $N_{\neg ant_\chi}$, with antecedent $ant_{\neg ant_\chi} = Nil$ and consequent $con_{\neg ant_\chi} = I_\omega$.
12. Set the node covered by the antecedent of $N_{\neg ant_\chi}$:

$$N_{\neg ant_{ant_\chi}} := Nil$$

13. Set the node covered by the negation of the antecedent of $N_{\neg ant_\chi}$:

$$N_{\neg ant_{\neg ant_\chi}} := Nil$$

Intuitively, the expand-worst-terminal-node operator attempts to increase the fitness of an individual, by partitioning the subspace covered by the worst terminal node (in terms of mean squared error) into two more subspaces. A high mean squared error is an indication of a poor function approximation. It is possible that by partitioning the subspace the accuracy of the function approximation can be increased. Thus, this operator caters for discontinuities in the input space.

- **Expand-any-terminal-node operator**: The expand-any-terminal-node operator partitions the sub-space of a random terminal node in an individual I_χ. The pseudo-code for the expand-any-terminal-node operator is identical to the expand-worst-terminal-node operator, except that step (1) should read:

1. Select N_χ from I_χ such that $con_\chi \neq Nil$.

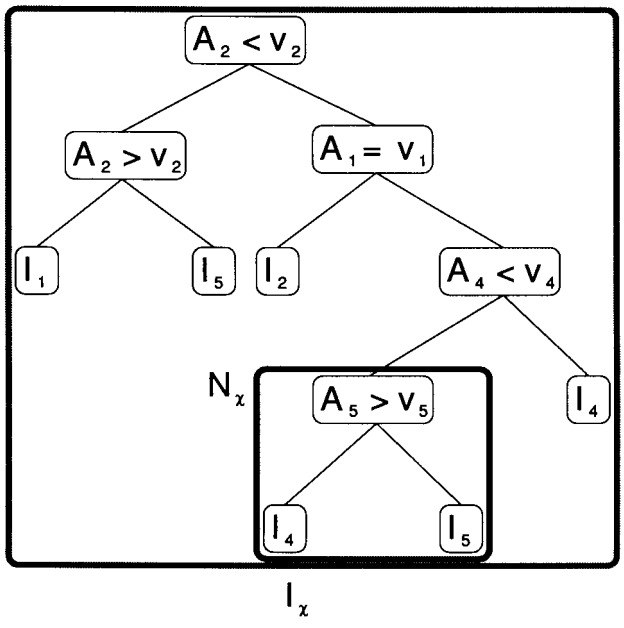

Figure 4.6: Illustration of the expand-worst-terminal-node operator for an individual I_χ

If the high mean squared error in the worst terminal node is due to a large variance in the data, the expand-worst-terminal-node operator will continually attempt to partition the subspace of the worse terminal node to no avail. This could lead to extremely slow convergence of the GPMCC method. The expand-any-terminal-node prevents this scenario from occurring by allowing any terminal node to be expanded.

- **Shrink operator**: The shrink operator replaces a non-terminal node of an individual I_χ with one of the non-terminal node's children. The pseudo-code for the shrink operator is as follows:

  1. Select N_χ (as shown in figure 4.7) from I_χ such that $(ant_\chi \neq Nil)$.
  2. If $\mathrm{U}(0,1) < 0.5$ then the current node becomes the node covered by the antecedent (the left node) $N_\chi := N_{ant_\chi}$.
  3. Otherwise, the current node becomes the node covered by the negation of the antecedent (the right node) $N_\chi := N_{\neg ant_\chi}$ (as shown in figure 4.7).

The shrink operator is responsible for removing introns from an individual, which is necessary to prevent code bloat.

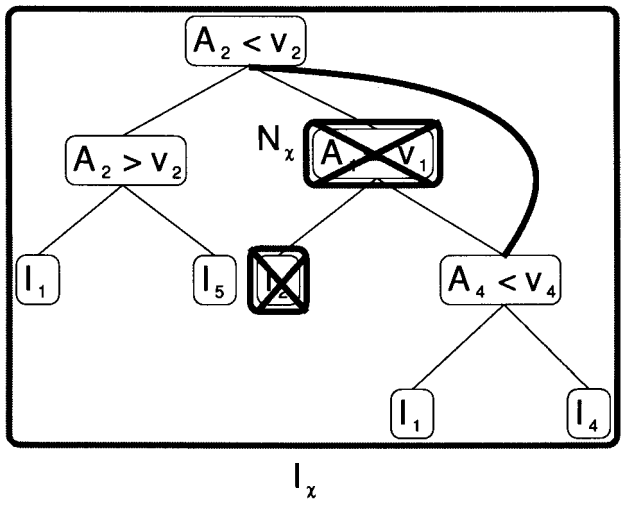

Figure 4.7: Illustration of the shrink operator for an individual I_χ

- **Perturb-worst-non-terminal-node operator**: The perturb-worst-non-terminal-node operator selects and perturbs a non-terminal node which has a higher relative error than all other non-terminal nodes in an individual I_χ. Once again, the relative error is determined using the mean squared error E_{MS} on the training set. This operator gives the GPMCC method an opportunity to optimise the partitions described by the non-terminal nodes of an individual. The pseudo-code for the perturb-worst-non-terminal-node operator is as follows:

  1. Select N_χ (as shown in figure 4.8) such that $\forall N_i \in I_\chi : (ant_\chi \neq Nil) \wedge (\mathrm{E_{MS}(N_\chi)} \leq \mathrm{E_{MS}(N_i)})$.
  2. If $\mathrm{U}(0,1) < \mathrm{U}_1$, where $U_1 \in [0,1]$ is a user-defined parameter
     (a) If A_ξ is a continuous-valued attribute
        i. If $\mathrm{U}(0,1) < \mathrm{U}_2$ where $U_2 \in [0,1]$ is a user-defined parameter, select an operator $\mathrm{op}(\xi) \in \{<, >, =, \neq\}$ (as shown in figure 4.8).

ii. Otherwise, adjust the attribute value v_ξ according to a Gaussian distribution $v_\xi := v_\xi + \frac{-(max-min)\mathrm{U}(0,1)^2}{2U_3(0.3)^2}$, where $\forall a_{\xi,i}, i = 1,\cdots I : (max \geq a_{\xi,i}) \wedge (min \leq a_{\xi,i})$, $U_3 \in \Re$ is a user-defined parameter, *min* is the minimum value for an attribute A_ξ and *max* is the maximum value for an attribute A_ξ. The standard deviation 0.3 of the Gaussian distribution provides an even distribution of the Gaussian function in the domain $[0, 1]$.

(b) Otherwise,

i. Randomly, select an attribute value $a_{\xi,i}$ for attribute A_ξ from a training pattern i, and let $v_\xi = a_{\xi,i}, i \in \{1,\cdots,|P|\}$.

3. Otherwise,

(a) Select an attribute $A_\xi, \xi = 1,\cdots,I$ from the attribute space of size I.

(b) If A_ξ is a continuous-valued attribute, select an operator $\mathrm{op}(\xi) \in \{<,>,=,\neq\}$.

(c) Otherwise, $\mathrm{op}(\xi) \in \{=\}$.

(d) Randomly select an attribute value $a_{\xi,i}$ for attribute A_ξ from a training pattern i, such that $v_\xi = a_{\xi,i}, i \in \{1,\cdots,|P|\}$.

4. Set the antecedent ant_χ of node N_χ to $(A_\xi\ \mathrm{op}(\xi)\ \mathrm{v}_\xi)$.

Intuitively, the partition described by a non-terminal node of an individual may not correctly partition the subspace *e.g.* if a test should have been $A_1 < 5$, but is actually $A_1 < 4.6$. The perturb-worst-non-terminal-node operator specifically attempts to adjust the test described by the worst non-terminal node (indicated by the largest mean squared error).

- **Perturb-any-non-terminal-node operator**: The perturb-any-non-terminal-node operator selects and perturbs a non-terminal node in an individual I_χ. The perturb-any-non-terminal-node operator is identical to the perturb-worst-non-terminal-node operator except that step (1) should read:

1. Select N_χ such that $(ant_\chi \neq Nil)$.

This operator allows for the perturbation of any non-terminal node, in order to prevent slow convergence in the case that the high mean squared error of the worst non-terminal node is due to high variation in the dataset.

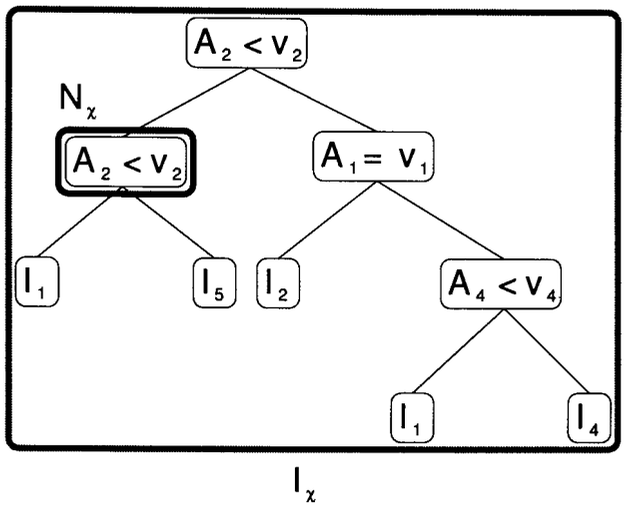

Figure 4.8: Illustration of the perturb-worst-non-terminal-node operator for an individual I_χ

- **Perturb-worst-terminal-node operator**: The perturb-worst-terminal-node operator selects and perturbs a terminal node which has a higher relative error than all other non-terminal nodes in an individual I_χ. This operator gives the GPMCC method an opportunity to optimise the non-linear approximations for the sub-space covered by the training and validation patterns described by the path to a terminal node. The perturb-worst-terminal-node operator is as follows:

  1. Select N_χ (as shown in figure 4.9) such that $\forall N_i \in I_\chi : (con_\chi \neq Nil) \wedge (\mathrm{E_{MS}(N_\chi)} \leq \mathrm{E_{MS}(N_i)})$.
  2. Select an individual I_ω from the fragment pool.
  3. Set the consequent con_χ of node N_χ to I_ω.

  Intuitively, the model described by the terminal node of an individual may be a poor fit of the data covered by the path of that terminal node. The perturb-worst-terminal-node operator randomly selects a new individual from the fragment pool to replace the current model.

- **Perturb-any-terminal-node operator**: The perturb-any-terminal-node operator selects and perturbs a terminal node in an individual I_χ. The perturb-any-terminal-node operator

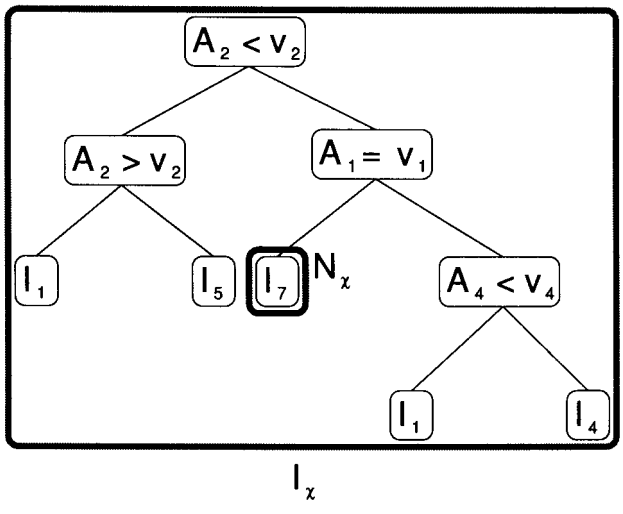

Figure 4.9: Illustration of the perturb-worst-terminal-node operator for an individual I_χ

is identical to the perturb-worst-terminal-node operator except that step (1) should read:

1. Select N_χ such that $(con_\chi \neq Nil)$.

This operator allows for the perturbation of any terminal node, in order to prevent slow convergence in the case that the high mean squared error of the worst terminal node is due to high variation in the dataset.

- **Reinitialise operator**: The reinitialise operator is a re-invocation of the initialisation operator.

Crossover operator

The crossover operator implements a standard genetic program crossover strategy. Two individuals (I_α and I_β) are chosen by tournament selection from the population and a crossover point is chosen for each individual. The two crossover points are spliced together to create a new individual I_γ. The pseudo-code for the crossover operator is as follows:

1. Select an individual $I_\alpha \in G_{GP}$ from the population G_{GP} (as shown in figure 4.10).

2. Select an individual $I_\beta \in G_{GP}$ from the population G_{GP} (as shown in figure 4.10).

3. Select a non-terminal node N_α from I_α (as shown in figure 4.10).

4. Select a node N_β from I_β (as shown in figure 4.10).

5. Create a new individual I_γ from I_α and I_β by installing N_β as a child of N_α.

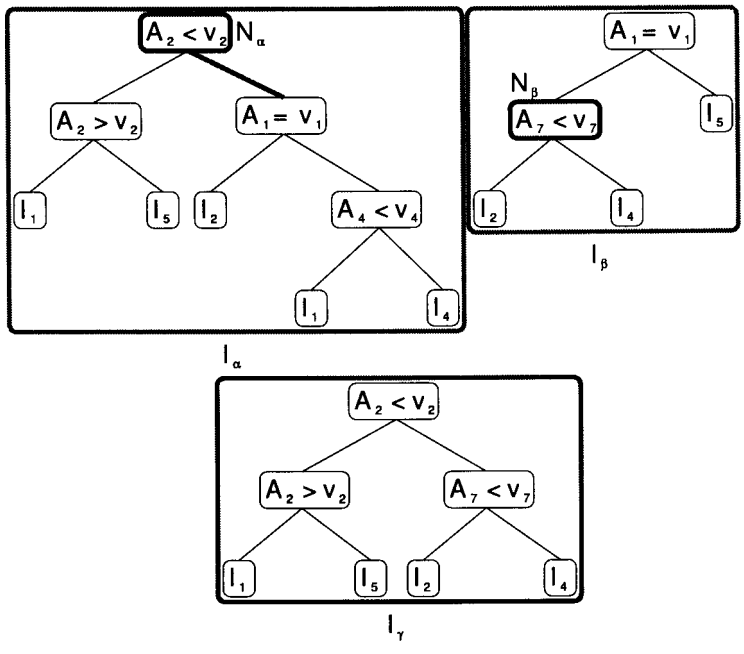

Figure 4.10: Illustration of the crossover operator for individuals I_α, I_β and I_γ

Fitness function

Like all evolutionary computing paradigms, the fitness function is the most important aspect of a genetic program, in that it serves to direct the algorithm toward optimal solutions. The fitness function used by the genetic program is an extended form of the *adjusted coefficient of determination* (the GASOPE fitness function) from equations (3.15) and (3.16):

$$R_a^2 = 1 - \frac{\sum_{i=1}^{s}(b_i - b^{\bullet}_{I_\chi,i})^2}{\sum_{i=1}^{s}(b_i - \overline{b}_i)^2} \cdot \frac{s-1}{s-k} \tag{4.4}$$

where s is the size of the sample set, b_i is the actual output of pattern i, $b^{\bullet}_{I_\chi,i}$ is the predicted output of individual I_χ for pattern i, and the model complexity d is calculated as follows:

$$d = \sum_{\mu=1}^{|I_\chi|} \begin{cases} 1 + \sum_{\xi=1}^{|I_\omega|} \sum_{\tau=1}^{|T_\xi|} \lambda_{\xi,\tau} & \text{if} \quad con_\mu \neq Nil \\ 1 & \text{if} \quad con_\mu = Nil \end{cases} \tag{4.5}$$

where I_χ is an individual in the set G_{GP} of individuals, $|I_\chi|$ is the number of nodes in I_χ, I_ω is the model at a terminal node, T_ξ is a term of I_ω and $\lambda_{\xi,\tau}$ is the order of term T_ξ. This fitness function penalises the complexity of an individual I_χ by penalising the size of an individual and the complexity of each of the non-terminal nodes of that individual, *i.e.* the number of nodes and the complexity of each leaf node as shown equation (3.16).

4.4 EXPERIMENTAL RESULTS

This section discusses the experimental procedure and results of the GPMCC method *vs.* NeuroLinear and Cubist, applied to various data sets obtained from the UCI machine learning repository and a number of artificially created datasets. Section 4.4.1 presents the various data sets. The influence of a number of key GPMCC parameters are discussed in section 4.4.2. Section 4.4.3 presents the experimental results for the functions listed in section 4.4.1. The quality of the generated rules is discussed in section 4.4.4.

4.4.1 Datasets

The GPMCC method was evaluated on a number of benchmark approximation databases from the machine learning repository as well as a number of artificial databases [14]. Table 4.1 describes each of the UCI databases used in this thesis. The artificial databases used in this thesis were created to analyse various approximation problems not sufficiently covered by the UCI machine learning repository, *e.g.* time-series, and to provide a number of large databases, exceeding 10000 patterns, with which to analyse the performance of the GPMCC method.

The *House-16H* data set in table 4.1 is a particularly difficult problem. Apart from being large, the data set also has very large values for its attributes. The performance in terms of rule accuracy of the various data mining algorithms used later in this thesis is expected to be poor, as is shown by this and the next section. Because the adjusted coefficient of determination is

Table 4.1: Databases obtained from the UCI machine learning repository (Attributes: N = Nominal, C = Continuous)

Dataset	Samples	Attributes	Prediction task
Abalone	4177	1N, 7C	Age of abalone specimens
Auto-mpg	392	7C	Car fuel consumption in miles per gallon
Elevators	16599	18C	Action taken for controlling an F16 aircraft
Federal Reserve Economic Data	1049	16C	1-Month credit deficit rate
House-16H	22784	16C	Median value of homes in 50 US states
Housing	506	13C	Median value of homes in Boston suburbs
Machine	209	6C	Relative CPU performance
Servo	167	2N, 2C	Response time of a servo mechanism

used as a fitness function for the GPMCC method, the scaling of attributes will not improve the accuracy of the GPMCC method.

The function describing the *Machine* data set in table 4.1 is piecewise linear. Therefore, the GPMCC method is not expected to perform substantially better, in terms of the number of rules generated, than other methods.

In addition to the problems listed in table 4.1, the following problems were also used:

- The **function example** database represents a discontinuous function, defined by a number of standard polynomial expressions:

$$y = \mathrm{U}(-1,1) + \begin{cases} 1.5x^2 - 7x + 2 & \text{if } (type = \text{'C'}) \wedge (x > 1) \\ x^3 + 400 & \text{if } (type = \text{'C'}) \wedge (x \leq 1) \\ 2x & \text{if } (type = \text{'A'}) \\ -2x & \text{if } (type = \text{'B'}) \end{cases}$$

  where $x \in [-10, 10]$ and $type \in \{\text{'A'}, \text{'B'}, \text{'C'}\}$. The database consists of 1000 patterns, with 3 attributes per pattern.

- The **Lena Image** database represents a 128×128 grey-scale version of the famous "Lena" image, which is used for comparing different image compression techniques [88]. The data consists of 11 continuous valued attributes, which represents a context (or footprint) for a given pixel. The objective of an approximation method is to infer

rules between the context pixels and the target pixel. The database is large and consists of 16384 patterns.

- The **Mono Sample** database represents an approximately 4 second long sound clip, sampled in mono at 8000 hertz (the chorus of U2's "Pride (In the name of love)"). The database consists of 31884 patterns, with 5 attributes per pattern. The objective of an approximation method is to infer rules between a sample and a context of previous samples.

- The **Stereo Sample** database represents an approximately 4 second long sound clip, sampled in stereo at 8000 hertz (the chorus of U2's "Pride (In the name of love)"). The database consists of 31430 patterns, with 9 attributes per pattern. The objective of an approximation method is to infer rules between a sample in the left channel and a number of sample points in the left and right channels.

- The **Time-series** database represents a discontinuous application of components of the Rossler and Lorenz attractors (from section 3.4.1), the Henon map and a polynomial term:

$$w_{n+1} = x_n y_n z_n$$

$$x_{n+1} = 1 - a \cdot x_n^2 + b \cdot y_n$$

$$y_{n+1} = x_n - a \cdot y_n$$

$$z_{n+1} = x_n y_n - b \cdot z_n$$

$$p = \mathrm{U}(0,1) + \begin{cases} x_{n+1} & \text{if} \quad (x_n \geq 0) \\ y_{n+1} & \text{if} \quad (x_n < 0) \wedge (y_n \geq 0) \\ z_{n+1} & \text{if} \quad (x_n < 0) \wedge (y_n < 0) \wedge (z_n \geq 0) \\ w_{n+1} & \text{if} \quad (x_n < 0) \wedge (y_n < 0) \wedge (z_n < 0) \end{cases}$$

where $x_0, y_0, z_0, w_0 \sim U(-5,5)$. The database consists of 1000 patterns, with 4 attributes per pattern, generated using the Runga-Kutta method with order 4 [16].

4.4.2 Parameter influence

This section discusses the key parameters of the GPMCC method. These key parameters are

- the **fragment lifetime**, which controls how long unused fragments remain in the fragment pool,
- the **leaf optimisation rate**, which controls how often the GASOPE method is called to obtain a model for a terminal node,
- **windowing**, which controls the iterative learning sample size,
- and **fragment pool initialisation**, which controls the initial terminal set of the GPMCC method.

The databases used to test the influence of key GPMCC parameters were the artificial databases of the previous section. The artificial databases were used because they are well defined, easily understandable and diverse. Four databases from the UCI machine learning repository have been selected, *i.e.* Abalone, Elevators, Federal Reserve Economic Data and House-16H, to provide additional problem diversity.

Each of the databases was split up into a training set, a validation set and a generalisation set. The training set was used to train the GPMCC method, the validation set was used to validate the models of the GASOPE method and the generalisation set was used to test the performance of the GPMCC method on unseen data. The training set for each database consisted of roughly 80% of the patterns, with the remainder of the patterns split evenly among the validation and generalisation sets (80% : 10% : 10%). The GPMCC initialisation for each of the databases is shown by table 4.2. For all datasets the maximum polynomial order was set to 5, except for the house-16H dataset, for which the maximum polynomial order was set to 10.

Generally speaking, the parameters prefixed by "Function" in table 4.2 have the same meaning as that of section 3.4.2 and control the behaviour of the model optimisation algorithm of the fragment pool. These parameters are soft options for the GPMCC method, because they are automatically adjusted if they violate any of the restrictions of section 3.4.2. The parameters prefixed by "Decision" control the behaviour of the genetic program for generating model trees. In general, "consequent" refers to terminal nodes and "antecedent" refers to non-terminal nodes. The mnemonic "CA" stands for continuous antecedent, "NA" stands for nominal antecedent, "ME" stands for mutate expand and "MN" stands for mutate node (perturb).

Table 4.2: GPMCC initialisation parameters

Parameter	Value
SyntaxMode	1
Clusters	30
ClusterEpochs	10
FunctionMutationRate	0.1
FunctionCrossoverRate	0.2
FunctionGenerations	100
FunctionIndividuals	30
PolynomialOrder	5
FunctionPercentageSampleSize	0.01
FunctionMaximumComponents	10
FunctionElite	0.1
FunctionCutOff	0.001
DecisionMaxNodes	30
DecisionMEWorstVsAnyConsequent	0.5
DecisionMECreateVsRedistributeLeafNodes	0.5
DecisionMNAntecedentVsConsequent	0.5
DecisionMNWorstVsAnyAntecedent	0.5
DecisionMNWorstVsAnyConsequent	0.5
DecisionReoptimizeVsSelectLeaf	0.1
DecisionMutateExpand	0.3
DecisionMutateShrink	0.3
DecisionMutateNode	0.3
DecisionMutateReinitialize	0.1
DecisionNAAttributeVsClassOptimize	0.2
DecisionCAAttributeOptimize	0.1
DecisionCAClassOptimize	0.6
DecisionCAConditionOptimize	0.3
DecisionCAClassVsGaussian	0.1
DecisionCAClassPartition	0.1
DecisionCAConditionalPartition	0.1
DecisionPoolNoClustersStart	30
DecisionPoolNoClustersDivision	2
DecisionPoolNoClusterEpochs	1000
DecisionPoolFragmentLifeTime	50
DecisionInitialPercentageSampleSize	0.1
DecisionSampleAcceleration	0.005
DecisionNoIndividuals	100
DecisionNoGenerations	10
DecisionElite	0.0
DecisionMutationRateInitial	0.2
DecisionMutationRateIncrement	0.01
DecisionMutationRateMax	0.6
DecisionCrossoverRate	0.1
CrossValidation	0

The GPMCC method utilises a variable mutation rate. The mutation rate is initially set to a default parameter ("DecisionMutationRateInitial"). Every time the accuracy of the best individual in a generation does not increase, the mutation rate is increased (by the amount specified by "DecisionMutationRateIncrement") up to a maximum mutation rate (given by "DecisionMutationRateMax").

This variable mutation rate helps to prevent stagnation. If the accuracy of the best individual does not improve over a number of generations, the increase in mutation rate injects more new genetic material into the population. However, if the accuracy of the best individual does improve, then the mutation rate is reset to the initial mutation rate.

A dagger, †, on the right hand side of results listed in tables 4.3 to 4.28 indicate the best result for a particular experiment set. In the event of ties between results, two other types of daggers are used: $\dagger^{\dagger}$ indicates that a particular result was judged the best of the experiment set using the adjusted coefficient of determination, and $\dagger^{\diamond}$ indicates that a particular result was judged the best of the experiment set using the mean squared error.

Fragment lifetime

The fragment lifetime ("DecisionPoolFragmentLifetime" in table 4.2) controls how long an unused GASOPE fragment remains in the fragment pool. Intuitively, a longer fragment lifetime results in a larger memory overhead to store unused fragments. A larger pool may result in slower convergence due to the increased number of fragments that could be selected at a leaf node. A smaller pool could result in sub-optimal convergence due to over-fitting.

Tables 4.3 - 4.7 show the effect of the fragment lifetime on the outcomes of the GPMCC method. Generally speaking, the fragment lifetime has a definite effect on the average simulation completion time $\bar{t}$. A decrease in the fragment lifetime results in a decrease in the average simulation completion time. The fragment lifetime has no significant effect on any of the other outcomes (MSE, number of nodes, R^2, R_a^2 *etc.*).

A fragment lifetime of 5 seems to result in over-fitting for some of the databases. *Abalone*, *House-16H*, and *Function Example* (shown in table 4.3) show a significant increase in both the average generalisation mean squared error $\overline{GMSE}$ and the standard deviation of the mean squared error σ_{GMSE}. Even though the fragment lifetime has no significant effect on generalisation accuracy, four of the nine datasets obtained their best generalisation performance with

Table 4.3: GPMCC method: Fragment lifetime 5

Dataset	$\overline{TMSE}$ $\overline{R^2_T}$ $\overline{R^2_{aT}}$ $\overline{Nodes}$		σ_{TMSE} $\sigma_{R^2_T}$ $\sigma_{R^2_{aT}}$ σ_{Nodes}	$\overline{GMSE}$ $\overline{R^2_G}$ $\overline{R^2_{aG}}$ $\bar{t}$		σ_{GMSE} $\sigma_{R^2_G}$ $\sigma_{R^2_{aG}}$ $\frac{\overline{TMSE}}{\overline{GMSE}}$	
Abalone	4.5854		0.0275	7.3979		10.1001	
	0.5614		0.0026	0.5126		0.1065	
	0.5571		0.0025	0.4765		0.1009	
	4.0000		1.4622	1264.3700	†	0.6198	
Elevators	0.0000	††	0.0000	0.0000	††	0.0000	
	0.8832	†	0.0065	0.8796	†	0.0076	
	0.8822	†	0.0063	0.8705	†	0.0078	
	7.8667		2.2087	1873.5300	†	1.0000	††
Federal	0.0441	†	0.0032	0.0521		0.0189	
Reserve	0.9961	†	0.0003	0.9954		0.0017	
Economic	0.9960	†	0.0003	0.9928		0.0027	
Data	5.5333		1.7367	636.3330	†	0.8477	
Function	0.4687		1.4012	0.4804		1.4807	
Example	1.0000		0.0000	1.0000		0.0001	
	1.0000		0.0000	1.0000		0.0001	
	7.8000		1.4480	953.1670	†	0.9757	
House-16H	1691964000.0000	†	63812800.0000	2320560000.0000		2949560000.0000	
	0.3916	†	0.0229	0.3606		0.0743	
	0.3887	†	0.0224	0.3375		0.0699	
	11.0667		3.3418	2940.3000	†	0.7291	
Lena	121.2942		3.9576	127.1936		6.7631	
Image	0.9402		0.0020	0.9377		0.0033	
	0.9398		0.0019	0.9348		0.0035	
	13.8000		3.7729	1671.2700	†	0.9536	†
Mono	301.4220		6.7250	302.9520		8.6692	
Sample	0.6873		0.0070	0.6799		0.0092	
	0.6868		0.0070	0.6761		0.0093	
	10.2000		2.0745	2437.8300	†	0.9950	
Stereo	200.4300	†	1.0665	200.3480	†	1.1636	
Sample	0.8241	†	0.0009	0.8191	†	0.0011	
	0.8239	†	0.0009	0.8174	†	0.0010	
	5.1333		1.2794	4584.1300	†	1.0004	†
Time-	1.1065	†	1.7963	1.0884		1.6558	
series	0.9912	†	0.0143	0.9894		0.0162	
	0.9905	†	0.0155	0.9396		0.1840	
	7.0000		1.8937	687.4000	†	1.0167	

Table 4.4: GPMCC method: Fragment lifetime 50

Dataset	$\overline{TMSE}$ $\overline{R^2_T}$ $\overline{R^2_{aT}}$ $\overline{Nodes}$	σ_{TMSE} $\sigma_{R^2_T}$ $\sigma_{R^2_{aT}}$ σ_{Nodes}	$\overline{GMSE}$ $\overline{R^2_G}$ $\overline{R^2_{aG}}$ $\bar{t}$	σ_{GMSE} $\sigma_{R^2_G}$ $\sigma_{R^2_{aG}}$ $\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5661	0.0304	5.1339 †	0.4080
	0.5632	0.0029	0.5660 †	0.0345
	0.5588	0.0028	0.5279 †	0.0390
	4.1333	1.2521	1376.5300	0.8894 †
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8811	0.0069	0.8771	0.0088
	0.8802	0.0068	0.8696	0.0084
	6.9333	2.1961	2045.0700	1.0000
Federal Reserve Economic Data	0.0450	0.0022	0.0477	0.0034
	0.9961	0.0002	0.9958	0.0003
	0.9959	0.0002	0.9938 †	0.0008
	5.3333	1.0613	743.3330	0.9418
Function Example	0.1004	0.0032	0.0915	0.0034
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.8667	1.3578	967.4000	1.0974
House-16H	1706552000.0000	68102200.0000	1775414000.0000 †	89054600.0000
	0.3864	0.0245	0.3755 †	0.0313
	0.3836	0.0243	0.3529 †	0.0330
	8.7333 †	2.4486	3442.2700	0.9612 †
Lena Image	119.9730 †	3.6204	125.8572 †	4.6576
	0.9408 †	0.0018	0.9383 †	0.0023
	0.9405 †	0.0017	0.9354 †	0.0025
	13.3333	3.6040	1894.8000	0.9532
Mono Sample	301.6280	5.3602	302.7020	6.9376
	0.6871	0.0056	0.6802	0.0073
	0.6866	0.0055	0.6765	0.0073
	9.3333	2.1064	2740.8000	0.9965
Stereo Sample	201.4300	2.1446	201.5580	2.5642
	0.8233	0.0019	0.8180	0.0023
	0.8231	0.0019	0.8163	0.0024
	5.0000	1.7420	4675.4700	0.9994
Time-series	1.1449	1.4470	0.9168 †	1.0918
	0.9909	0.0115	0.9910 †	0.0107
	0.9902	0.0124	0.9691	0.0546
	7.2000	1.8458	795.3670	1.2489

Table 4.5: GPMCC method: Fragment lifetime 100

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5658	0.0355	16.0463	59.4182
	0.5632	0.0034	0.5418	0.1074
	0.5587	0.0032	0.5046	0.1009
	3.9333	1.1427	1471.9700	0.2846
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8827	0.0078	0.8776	0.0092
	0.8819	0.0076	0.8700	0.0081
	6.7333 †	2.3916	2062.2000	1.0000
Federal	0.0452	0.0019	0.0476 †	0.0051
Reserve	0.9961	0.0002	0.9958 †	0.0005
Economic	0.9959	0.0002	0.9936	0.0013
Data	5.1333	1.0417	845.3670	0.9501 †
Function	0.1001 †	0.0025	0.0908 †	0.0030
Example	1.0000 †°	0.0000	1.0000 †°	0.0000
	1.0000 †°	0.0000	1.0000 †°	0.0000
	7.4000 †	0.8137	1008.2700	1.1017 †
House-16H	1708182000.0000	59607800.0000	1816690000.0000	257592000.0000
	0.3858	0.0214	0.3642	0.0737
	0.3831	0.0214	0.3434	0.0707
	9.1333	3.0596	3571.0700	0.9403
Lena	121.5576	3.4550	127.4858	5.1076
Image	0.9400	0.0017	0.9375	0.0025
	0.9397	0.0017	0.9350	0.0027
	12.4000	3.3280	2024.3700	0.9535
Mono	300.2160	5.0925	300.9340	6.2382
Sample	0.6885	0.0053	0.6821	0.0066
	0.6881	0.0053	0.6783	0.0065
	10.0000	1.9476	2773.1000	0.9976
Stereo	201.3880	2.4986	201.5300	2.9388
Sample	0.8233	0.0022	0.8181	0.0027
	0.8231	0.0022	0.8165	0.0027
	4.6667	2.0398	4819.2300	0.9993
Time-	1.3721	1.4380	1.2856	1.2400
series	0.9891	0.0115	0.9874	0.0121
	0.9882	0.0124	0.9301	0.1829
	6.6667	1.4933	859.4000	1.0673

Table 4.6: GPMCC method: Fragment lifetime 200

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5606 †	0.0409	5.3216	0.8711
	0.5637 †	0.0039	0.5501	0.0736
	0.5592 †	0.0034	0.5088	0.0866
	4.2000	1.6274	1483.3700	0.8570
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8823	0.0054	0.8777	0.0068
	0.8813	0.0053	0.8693	0.0061
	7.6667	2.1867	2117.2000	1.0000
Federal Reserve Economic Data	0.0447	0.0021	0.0591	0.0373
	0.9961	0.0002	0.9948	0.0033
	0.9959	0.0002	0.9919	0.0050
	5.4000	1.4288	826.8670	0.7555
Function Example	0.1005	0.0028	0.0919	0.0040
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.4667	1.2521	1147.8700	1.0933
House-16H	1730830000.0000	49356800.0000	1826146000.0000	124870200.0000
	0.3777	0.0177	0.3576	0.0439
	0.3750	0.0174	0.3351	0.0484
	9.0667	2.7029	3542.1300	0.9478
Lena Image	121.6324	3.4741	129.2850	9.8847
	0.9400	0.0017	0.9366	0.0048
	0.9397	0.0017	0.9339	0.0050
	12.2667	2.8031	2066.8000	0.9408
Mono Sample	300.9200	5.6947	302.1420	7.3790
	0.6878	0.0059	0.6808	0.0078
	0.6873	0.0059	0.6771	0.0077
	9.5333	2.0965	2845.7000	0.9960
Stereo Sample	202.2480	2.6624	203.5020	5.1323
	0.8225	0.0023	0.8163	0.0046
	0.8223	0.0023	0.8147	0.0048
	4.6667	1.4933	5001.7700	0.9938
Time-series	1.1618	1.4913	0.9247	1.2046
	0.9907	0.0119	0.9910	0.0118
	0.9901	0.0128	0.9612	0.1351
	6.7333	1.5522	915.8330	1.2564 †

Table 4.7: GPMCC method: Fragment lifetime 400

Dataset	$\overline{TMSE}$		σ_{TMSE}	$\overline{GMSE}$		σ_{GMSE}
	$\overline{R^2_T}$		$\sigma_{R^2_T}$	$\overline{R^2_G}$		$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$		$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$		$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$		σ_{Nodes}	$\bar{t}$		$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5682		0.0340	5.5379		1.6687
	0.5630		0.0033	0.5383		0.1071
	0.5588		0.0030	0.5023		0.1012
	3.6667	†	1.2130	1575.8000		0.8249
Elevators	0.0000		0.0000	0.0000		0.0000
	0.8819		0.0074	0.8782		0.0094
	0.8810		0.0072	0.8702		0.0086
	7.0667		2.4344	2172.7300		1.0000
Federal Reserve Economic Data	0.0449		0.0028	0.0518		0.0218
	0.9961		0.0002	0.9954		0.0019
	0.9959		0.0002	0.9932		0.0032
	5.0000	†	1.3896	904.9330		0.8667
Function Example	0.1004		0.0030	0.0912		0.0036
	1.0000		0.0000	1.0000		0.0000
	1.0000		0.0000	1.0000		0.0000
	7.4000		1.1017	1198.7000		1.1007
House-16H	1705792000.0000		65710600.0000	1856478000.0000		390526000.0000
	0.3867		0.0236	0.3592		0.0728
	0.3839		0.0230	0.3374		0.0685
	9.5333		3.5982	3578.9000		0.9188
Lena Image	122.2352		3.7588	130.0164		9.8415
	0.9397		0.0019	0.9363		0.0048
	0.9394		0.0018	0.9336		0.0051
	12.1333	†	3.2242	2017.7300		0.9402
Mono Sample	299.7100	†	4.0271	300.2380	†	4.6945
	0.6890	†	0.0042	0.6828	†	0.0050
	0.6886	†	0.0042	0.6792	†	0.0050
	9.2667	†	1.5522	2799.3000		0.9982 †
Stereo Sample	200.5820		1.7623	200.5640		2.1152
	0.8240		0.0015	0.8189		0.0019
	0.8238		0.0015	0.8173		0.0020
	4.6667	†°	1.8257	5066.8300		1.0001
Time-series	1.3141		1.5510	1.1545		1.1890
	0.9895		0.0124	0.9887		0.0116
	0.9889		0.0130	0.9740	†	0.0307
	6.2667	†	1.8557	982.1000		1.1382

a fragment lifetime of 50. Additionally, eight of the datasets where distributed between a fragment lifetime of 5 and 100. Therefore, a fragment lifetime of 50 appears to be the best choice for this parameter, since it is both consistent and computationally less expensive than the larger fragment lifetimes.

Leaf optimisation rate

The leaf optimisation rate ("DecisionReoptimizeVsSelectLeaf" in table 4.2) controls the rate at which a leaf node is optimised using the GASOPE method. The GASOPE method takes ± 1.5 seconds to perform an optimisation (refer to section 3.4.3). A larger leaf optimisation rate should thus result in the GPMCC method taking longer to complete each simulation.

Tables 4.8 - 4.11 show the effect of the leaf optimisation rate on the outcomes of the GPMCC method. As expected, an increase in the leaf optimisation rate resulted in an increase in the average simulation completion time $\bar{t}$. An increase in the leaf optimisation rate also resulted in an increase in training accuracy ($\overline{TMSE}$, R^2_T and R^2_{aT}). However, there was no significant increase in generalisation accuracy ($\overline{GMSE}$, R^2_G and R^2_{aG}), except in the case of the *Abalone* data set where the generalisation accuracy actually decreased. In table 4.11, overfitting was observed in some instances (*Abalone*, *Federal Reserve Economic Data* and *Time-series*), but not in others (*House-16H*, *Function Example*, *Lena Image*, *Mono Sample* and *Stereo Sample*).

Interestingly, seven of the nine databases achieved the best training-generalisation ratio $\frac{\overline{TMSE}}{\overline{GMSE}}$ for a leaf optimisation value of 0.05 (shown in table 4.8). This clearly shows that the generalisation accuracy is not significantly affected by an increase in the leaf optimisation rate. A low leaf optimisation rate is clearly desired, because the performance gain in terms of the average simulation completion time outweighs any increase in training performance, particularly if there was no significant increase in generalisation performance. For this reason, a leaf optimisation rate of 0.05 appears to be the best choice for this parameter.

Windowing

Windowing is controlled by two parameters; the initial window size as a percentage of the total number of training patterns ("DecisionInitialPercentageSampleSize" in table 4.2) and the acceleration rate by which patterns are injected into the window ("DecisionSampleAccelera-

Table 4.8: GPMCC method: Leaf optimisation rate 0.05

Dataset	$\overline{TMSE}$		σ_{TMSE}	$\overline{GMSE}$		σ_{GMSE}	
	$\overline{R^2_T}$		$\sigma_{R^2_T}$	$\overline{R^2_G}$		$\sigma_{R^2_G}$	
	$\overline{R^2_{aT}}$		$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$		$\sigma_{R^2_{aG}}$	
	$\overline{Nodes}$		σ_{Nodes}	$\bar{l}$		$\frac{\overline{TMSE}}{\overline{GMSE}}$	
Abalone	4.5836		0.0308	6.4099		6.7010	
	0.5615		0.0029	0.5426		0.1079	
	0.5572		0.0028	0.5071		0.1027	
	4.0000		1.5536	991.6330	†	0.7151	
Elevators	0.0000		0.0000	0.0000		0.0000	
	0.8805		0.0062	0.8783		0.0084	
	0.8797		0.0061	0.8709		0.0077	
	6.5333		1.6344	1648.6700	†	1.0000	††
Federal Reserve Economic Data	0.0452		0.0034	0.0480	†	0.0068	
	0.9960		0.0003	0.9957	†	0.0006	
	0.9959		0.0003	0.9935	†	0.0014	
	5.0000	†	1.2865	485.3330	†	0.9428	†
Function Example	0.1000		0.0034	0.0910		0.0035	
	1.0000		0.0000	1.0000		0.0000	
	1.0000		0.0000	1.0000		0.0000	
	7.6000		1.1919	539.4330	†	1.0997	†
House-16H	1698136000.0000		68116800.0000	1807944000.0000		113766200.0000	
	0.3894		0.0245	0.3640		0.0400	
	0.3865		0.0242	0.3395		0.0463	
	10.2667		3.7318	2762.8700	†	0.9393	
Lena Image	122.2988		3.4490	128.1206		4.3605	
	0.9397		0.0017	0.9372		0.0021	
	0.9394		0.0017	0.9346		0.0012	
	12.8000	†	3.2947	1681.1700	†	0.9546	†
Mono Sample	301.5540		6.3549	302.2460		8.1705	
	0.6871		0.0066	0.6807		0.0086	
	0.6867		0.0066	0.6769		0.0086	
	9.6667		2.1227	2414.7300	†	0.9977	†
Stereo Sample	201.3500		1.6139	201.4480		2.1278	
	0.8233		0.0014	0.8181		0.0019	
	0.8231		0.0014	0.8165		0.0021	
	4.9333		1.7006	3953.2300	†	0.9995	†
Time-series	1.4509		2.0606	1.0484		1.4091	
	0.9884		0.0164	0.9897		0.0138	
	0.9874		0.0181	0.8855		0.3011	
	6.4000	†	1.4044	474.0330	†	1.3840	†

Table 4.9: GPMCC method: Leaf optimisation rate 0.1

Dataset	$\overline{TMSE}$ $\overline{R^2_T}$ $\overline{R^2_{aT}}$ $\overline{Nodes}$	σ_{TMSE} $\sigma_{R^2_T}$ $\sigma_{R^2_{aT}}$ σ_{Nodes}	$\overline{GMSE}$ $\overline{R^2_G}$ $\overline{R^2_{aG}}$ $\bar{t}$	σ_{GMSE} $\sigma_{R^2_G}$ $\sigma_{R^2_{aG}}$ $\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5705	0.0355	5.2330 †	0.6863
	0.5628	0.0034	0.5576 †	0.0580
	0.5580	0.0033	0.5156 †	0.0680
	4.2000	1.7100	1436.9000	0.8734 †
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8799	0.0053	0.8748	0.0050
	0.8790	0.0051	0.8674	0.0048
	6.3333 †	2.1867	2128.3300	1.0000
Federal Reserve Economic Data	0.0447	0.0029	0.0485	0.0055
	0.9961	0.0003	0.9957	0.0005
	0.9959	0.0003	0.9932	0.0013
	5.7333	1.3374	794.2670	0.9223
Function Example	0.0997	0.0035	0.0915	0.0040
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.7333	1.4368	996.5330	1.0918
House-16H	1715448000.0000	61350600.0000	1787912000.0000	90663800.0000
	0.3832	0.0221	0.3711	0.0319
	0.3807	0.0218	0.3511	0.0314
	8.6000 †	2.5407	3481.7300	0.9595 †
Lena Image	120.0606	3.4722	125.8182	5.7197
	0.9408	0.0017	0.9383	0.0028
	0.9404	0.0017	0.9354	0.0028
	13.0667	3.5422	2017.5700	0.9542
Mono Sample	301.0520	5.7343	302.2340	7.2463
	0.6877	0.0060	0.6807	0.0077
	0.6872	0.0059	0.6771	0.0075
	8.8667 †	1.8889	2803.7000	0.9961
Stereo Sample	201.6580	2.9872	202.6600	5.4989
	0.8231	0.0026	0.8170	0.0050
	0.8229	0.0026	0.8153	0.0050
	5.0667	1.7006	4757.1300	0.9951
Time-series	1.4052	1.9056	1.0292 †	1.3389
	0.9888	0.0152	0.9899 †	0.0131
	0.9880	0.0164	0.9694 †	0.0696
	6.7333	2.0833	819.9670	1.3654

Table 4.10: GPMCC method: Leaf optimisation rate 0.2

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5632	0.0300	6.8353	8.6463
	0.5635	0.0029	0.5367	0.1142
	0.5592	0.0025	0.5010	0.1089
	3.7333 †	1.2299	2480.1300	0.6676
Elevators	0.0000	0.0000	0.0000 ††	0.0000
	0.8847	0.0071	0.8809 †	0.0089
	0.8837	0.0068	0.8723 †	0.0078
	7.9333	3.0050	2898.1700	1.0000
Federal Reserve Economic Data	0.0443	0.0028	0.0496	0.0078
	0.9961	0.0002	0.9956	0.0007
	0.9960	0.0002	0.9934	0.0013
	5.3333	1.5830	1494.1000	0.8927
Function Example	0.0999	0.0031	0.0909	0.0038
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.4667 †	1.0080	2015.8300	1.0990
House-16H	1681026000.0000	63924800.0000	2019260000.0000	1377860000.0000
	0.3956	0.0230	0.3654	0.0762
	0.3929	0.0228	0.3437	0.0729
	8.7333	2.8154	4715.7700	0.8325
Lena Image	118.9782	3.0875	126.2072	5.8012
	0.9413	0.0015	0.9382	0.0028
	0.9410	0.0015	0.9353	0.0029
	12.9333	3.5809	2626.0700	0.9427
Mono Sample	300.5480	5.1244	301.2480	6.7504
	0.6882	0.0053	0.6817	0.0071
	0.6877	0.0053	0.6780	0.0072
	9.4000	1.8495	3468.6000	0.9977
Stereo Sample	201.2660	1.8135	201.4240	1.9861
	0.8234	0.0016	0.8182	0.0018
	0.8232	0.0016	0.8166	0.0019
	4.3333 †	1.2130	6513.7000	0.9992
Time-series	1.3655 †	1.6599	1.2416	1.2834
	0.9891 †	0.0132	0.9879	0.0126
	0.9883 †	0.0141	0.9682	0.0380
	6.4667	1.5698	1682.8300	1.0998

Table 4.11: GPMCC method: Leaf optimisation rate 0.4

Dataset	$\overline{TMSE}$		σ_{TMSE}	$\overline{GMSE}$		σ_{GMSE}
	$\overline{R^2_T}$		$\sigma_{R^2_T}$	$\overline{R^2_G}$		$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$		$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$		$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$		σ_{Nodes}	$\bar{t}$		$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5585	†	0.0221	25.8332		111.4494
	0.5640	†	0.0021	0.5183		0.1117
	0.5594	†	0.0018	0.4785		0.1070
	4.0000		1.1447	4367.6300		0.1765
Elevators	0.0000	††	0.0000	0.0000		0.0000
	0.8847	†	0.0058	0.8798		0.0077
	0.8837	†	0.0057	0.8707		0.0080
	7.6000		2.2376	5008.1700		1.0000
Federal	0.0427	†	0.0031	0.0500		0.0057
Reserve	0.9963	†	0.0003	0.9956		0.0005
Economic	0.9961	†	0.0003	0.9930		0.0013
Data	5.3333		1.5830	3115.8300		0.8537
Function	0.0988	†	0.0031	0.0904	†	0.0030
Example	1.0000	†°	0.0000	1.0000	†°	0.0000
	1.0000	†°	0.0000	1.0000	†°	0.0000
	7.8667		1.3578	3809.7000		1.0925
House-16H	1651170000.0000	†	79038000.0000	1750206000.0000	†	119859600.0000
	0.4063	†	0.0284	0.3843	†	0.0422
	0.4034	†	0.0280	0.3596	†	0.0411
	10.3333		2.6436	6268.5300		0.9434
Lena	117.2918	†	4.8566	123.7744	†	7.4047
Image	0.9421	†	0.0024	0.9393	†	0.0036
	0.9418	†	0.0023	0.9361	†	0.0035
	14.7333		4.6307	3968.4000		0.9476
Mono	298.7060	†	3.8851	299.6060	†	5.1041
Sample	0.6901	†	0.0040	0.6835	†	0.0054
	0.6896	†	0.0041	0.6793	†	0.0059
	10.5333		2.5015	4645.1700		0.9970
Stereo	200.4060	†	0.8277	200.6060	†	0.8389
Sample	0.8242	†	0.0007	0.8189	†	0.0008
	0.8240	†	0.0007	0.8172	†	0.0006
	4.7333		1.7991	9672.4300		0.9990
Time-	1.4194		1.6810	1.3092		1.3016
series	0.9887		0.0134	0.9872		0.0127
	0.9880		0.0141	0.9740		0.0234
	5.8667		1.7953	3558.8300		1.0842

Table 4.12: GPMCC method: Initial window 0.05, window acceleration 0.005

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5765	0.0363	10.6855	25.6446
	0.5622	0.0035	0.5241	0.1451
	0.5581	0.0033	0.4907	0.1368
	3.5333 †	0.8996	1322.4300 †	0.4283
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8819	0.0060	0.8783	0.0075
	0.8809	0.0059	0.8701	0.0070
	7.1333	1.8144	1896.2700 †	1.0000
Federal Reserve Economic Data	0.0457	0.0026	0.0492	0.0078
	0.9960	0.0002	0.9956	0.0007
	0.9958	0.0002	0.9935	0.0013
	4.9333 †	1.4368	745.0670 †	0.9272 †
Function Example	0.1008	0.0027	0.0926	0.0033
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.3333 †	0.7581	836.9670 †	1.0886
House-16H	1719960000.0000	58444000.0000	2192540000.0000	1932294000.0000
	0.3816	0.0210	0.3427	0.0954
	0.3790	0.0207	0.3232	0.0903
	8.4667	3.0596	3128.1000 †	0.7845
Lena Image	121.0116	3.4832	126.7218	6.5353
	0.9403	0.0017	0.9379	0.0032
	0.9400	0.0017	0.9351	0.0032
	12.8667	3.9631	1798.3300 †	0.9549
Mono Sample	301.8660	6.6957	302.8780	8.5772
	0.6868	0.0070	0.6800	0.0091
	0.6864	0.0069	0.6765	0.0090
	9.2667 †	2.2118	2390.7300 †	0.9967 †
Stereo Sample	202.0540	2.9447	202.2560	3.6260
	0.8227	0.0026	0.8174	0.0033
	0.8225	0.0026	0.8160	0.0033
	4.1333 †	1.5477	4334.7700 †	0.9990
Time-series	1.4959	1.8987	1.1953	1.3302
	0.9881	0.0151	0.9883	0.0130
	0.9871	0.0164	0.9532	0.0883
	6.7333	1.7207	757.9330 †	1.2515 †

Table 4.13: GPMCC method: Initial window 0.05, window acceleration 0.01

Dataset	$\overline{TMSE}$ $\overline{R_T^2}$ $\overline{R_{aT}^2}$ $\overline{Nodes}$	σ_{TMSE} $\sigma_{R_T^2}$ $\sigma_{R_{aT}^2}$ σ_{Nodes}	$\overline{GMSE}$ $\overline{R_G^2}$ $\overline{R_{aG}^2}$ $\bar{t}$	σ_{GMSE} $\sigma_{R_G^2}$ $\sigma_{R_{aG}^2}$ $\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5640	0.0382	5.2366	0.6709
	0.5634	0.0037	0.5573	0.0567
	0.5589	0.0036	0.5192	0.0598
	4.2000	1.3493	1595.7700	0.8716
Elevators	0.0000 ††	0.0000	0.0000	0.0000
	0.8845 †	0.0084	0.8803	0.0104
	0.8836 †	0.0082	0.8726	0.0094
	7.0000	2.4635	2151.3000	1.0000
Federal Reserve Economic Data	0.0435	0.0025	0.0567	0.0264
	0.9962	0.0002	0.9950	0.0023
	0.9960	0.0002	0.9922	0.0033
	5.7333	1.3374	849.7670	0.7679
Function Example	0.0999	0.0033	0.0912	0.0041
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.6667	1.0933	1157.3300	1.0954
House-16H	1694530000.0000	77536200.0000	1791446000.0000	120896200.0000
	0.3907	0.0279	0.3698	0.0425
	0.3880	0.0274	0.3468	0.0403
	9.2667	3.8141	3746.1000	0.9459
Lena Image	119.4946	4.0812	127.8456	8.5962
	0.9411	0.0020	0.9374	0.0042
	0.9407	0.0019	0.9344	0.0043
	13.9333	3.9908	1990.3000	0.9347
Mono Sample	300.2680	5.3172	301.3240	6.7600
	0.6885	0.0055	0.6816	0.0071
	0.6880	0.0055	0.6779	0.0073
	9.7333	1.8557	2686.7700	0.9965
Stereo Sample	200.9620	1.1458	201.0280	1.6687
	0.8237	0.0010	0.8185	0.0015
	0.8235	0.0010	0.8170	0.0018
	4.4667	1.2794	5559.3700	0.9997
Time-series	0.9358	1.3316	0.7544	1.0153
	0.9925	0.0106	0.9926	0.0099
	0.9920	0.0114	0.9697	0.0792
	7.1333	1.8144	855.1000	1.2406

Table 4.14: GPMCC method: Initial window 0.05, window acceleration 0.02

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5545	0.0264	11.7964	35.9028
	0.5643	0.0025	0.5382	0.1113
	0.5596	0.0024	0.4987	0.1051
	4.2667	1.3374	1741.9000	0.3861
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8811	0.0078	0.8764	0.0085
	0.8803	0.0077	0.8687	0.0077
	7.2000	2.4269	2309.2300	1.0000
Federal Reserve Economic Data	0.0441	0.0024	0.0497	0.0094
	0.9961	0.0002	0.9956	0.0008
	0.9960	0.0002	0.9935	0.0014
	5.0667	1.6174	931.5000	0.8879
Function Example	0.1004	0.0026	0.0900	0.0031
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.7333	1.2299	1367.6000	1.1159
House-16H	1699556000.0000	63065400.0000	1897644000.0000	532012000.0000
	0.3889	0.0227	0.3536	0.0782
	0.3862	0.0221	0.3322	0.0744
	9.9333	3.0505	3856.1700	0.8956
Lena Image	119.3798	3.8368	125.1458	6.5550
	0.9411	0.0019	0.9387	0.0032
	0.9408	0.0019	0.9356	0.0034
	14.2667	4.1184	2085.0700	0.9539
Mono Sample	300.4640	5.1975	301.5520	6.2393
	0.6883	0.0054	0.6814	0.0066
	0.6878	0.0054	0.6774	0.0066
	10.6000	2.0611	2833.8700	0.9964
Stereo Sample	200.2760 †	1.3597	200.3160 †	1.3913
	0.8243 †	0.0012	0.8192 †	0.0013
	0.8241 †	0.0011	0.8175 †	0.0011
	4.8667	2.161310	5928.9700	0.9980
Time-series	1.1903	1.3186	1.0331	1.0850
	0.9905	0.0105	0.9899	0.0106
	0.9897	0.0117	0.9157	0.2131
	6.8667	1.2794	954.6000	1.1521

Table 4.15: GPMCC method: Initial window 0.05, window acceleration 0.04

Dataset	$\overline{TMSE}$		σ_{TMSE}	$\overline{GMSE}$		σ_{GMSE}
	$\overline{R^2_T}$		$\sigma_{R^2_T}$	$\overline{R^2_G}$		$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$		$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$		$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$		σ_{Nodes}	$\bar{t}$		$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5529	†	0.0370	5.1910		0.4104
	0.5645	†	0.0035	0.5611		0.0347
	0.5600	†	0.0030	0.5225		0.0339
	4.1333		1.2521	1749.6700		0.8771
Elevators	0.0000		0.0000	0.0000		0.0000
	0.8804		0.0078	0.8752		0.0087
	0.8796		0.0076	0.8678		0.0082
	6.6000	†	2.429700	2514.7700		1.0000
Federal	0.0436		0.0021	0.0522		0.0111
Reserve	0.9962		0.0002	0.9954		0.0010
Economic	0.9960		0.0002	0.9930		0.0017
Data	5.6667		1.4223	927.9000		0.8356
Function	0.1001		0.0033	0.0893	†	0.0046
Example	1.0000		0.0000	1.0000	†◇	0.0000
	1.0000		0.0000	1.0000	†◇	0.0000
	8.2000		1.7889	1308.5000		1.1204
House-16H	1687488000.0000		70731200.0000	1761580000.0000	†	86005600.0000
	0.3932		0.0254	0.3803	†	0.0303
	0.3906		0.0252	0.3594	†	0.0295
	10.0000		3.2270	3861.4000		0.9579
Lena	119.0428	†	4.1056	127.3112		8.0203
Image	0.9413	†	0.0020	0.9376		0.0039
	0.9409	†	0.0020	0.9347		0.0037
	13.8000		3.6237	2102.4700		0.9351
Mono	300.5620		5.7319	302.1180		7.1440
Sample	0.6882		0.0059	0.6808		0.0075
	0.6877		0.0059	0.6767		0.0077
	10.0000		2.6130	3106.6300		0.9948
Stereo	201.5460		3.0197	201.8520		3.7076
Sample	0.8232		0.0027	0.8178		0.0033
	0.8230		0.0026	0.8161		0.0033
	4.8000		1.4239	6203.1700		0.9985
Time-	0.4407	†	0.6411	0.3631	†	0.4587
series	0.9965	†	0.0051	0.9964	†	0.0045
	0.9963	†	0.0054	0.9913	†	0.0171
	7.1333		1.0417	904.2330		1.2137

Table 4.16: GPMCC method: Initial window 0.1, window acceleration 0.005

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5658	0.0338	10.1196	26.2766
	0.5632	0.0032	0.5315	0.1207
	0.5586	0.0030	0.4922	0.1191
	4.0000	1.1447	1455.6300	0.4512
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8834	0.0062	0.8790	0.0079
	0.8825	0.0060	0.8711	0.0077
	7.0667	1.9989	2100.6700	1.0000
Federal	0.0438	0.0026	0.0496	0.0068
Reserve	0.9962	0.0002	0.9956	0.0006
Economic	0.9960	0.0002	0.9933	0.0012
Data	5.5333	1.4794	771.4000	0.8815
Function	0.1004	0.0026	0.0915	0.0030
Example	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.5333	1.0417	1004.9700	1.0975
House-16H	1708888000.0000	74731200.0000	1769664000.0000	104933800.0000
	0.3855	0.0269	0.3775	0.0369
	0.3829	0.0267	0.3564	0.0364
	9.6000	2.3577	3297.7000	0.9657 †
Lena	121.3112	3.4365	126.8750	4.8966
Image	0.9402	0.0017	0.9378	0.0024
	0.9398	0.0017	0.9348	0.0024
	13.4667	3.8483	1951.9700 †	0.9561
Mono	301.8080	6.3805	303.3280	8.5357
Sample	0.6869	0.0066	0.6795	0.0090
	0.6864	0.0066	0.6758	0.0090
	9.5333	1.8889	2722.3000	0.9950
Stereo	200.8280	1.7459	200.9520	2.1067
Sample	0.8238	0.0015	0.8186	0.0019
	0.8236	0.0015	0.8171	0.0019
	4.4000	1.5888	4850.8300	0.9994
Time-	1.2285	1.5279	1.0462	1.2766
series	0.9902	0.0122	0.9898	0.0125
	0.9894	0.0133	0.9400	0.1847
	6.3333 †	1.3218	839.3330	1.1742

Table 4.17: GPMCC method: Initial window 0.1, window acceleration 0.01

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5643	0.0285	6.5100	7.0911
	0.5634	0.0027	0.5402	0.1087
	0.5588	0.0024	0.5019	0.1038
	4.0000	1.0171	1654.7300	0.7011
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8810	0.0074	0.8774	0.0094
	0.8801	0.0073	0.8698	0.0087
	6.7333	1.8742	2291.3000	1.0000
Federal	0.0431	0.0033	0.0482 †	0.0049
Reserve	0.9962	0.0003	0.9957 †	0.0004
Economic	0.9961	0.0003	0.9933 †	0.0010
Data	6.0000	1.2595	902.0670	0.8937
Function	2.3835	11.4882	0.3013	1.1123
Example	1.0000	0.0004	1.0000	0.0000
	1.0000	0.0004	1.0000	0.0000
	8.1333	1.5477	1228.1700	7.9112 †
House-16H	1705032000.0000	68749600.0000	1767764000.0000	90879600.0000
	0.3869	0.0247	0.3782	0.0320
	0.3842	0.0244	0.3558	0.0302
	9.3333	3.0663	3954.6000	0.9645
Lena	119.9782	4.1011	203.0900	403.0240
Image	0.9408	0.0020	0.9053	0.1711
	0.9405	0.0020	0.9025	0.1705
	12.5333 †	3.4314	2107.5000	0.5908
Mono	302.7440	6.6125	304.9620	8.3493
Sample	0.6859	0.0069	0.6778	0.0088
	0.6855	0.0068	0.6741	0.0086
	9.4000	2.5407	3180.5700	0.9927
Stereo	200.5580	1.0425	200.5260	1.2339
Sample	0.8240	0.0009	0.8190	0.0011
	0.8238	0.0009	0.8173	0.0012
	5.0667	1.6174	5454.8000	1.0002 †
Time-	0.9743	1.3699	0.9567	1.3130
series	0.9922	0.0109	0.9906	0.0128
	0.9917	0.0116	0.9761	0.0315
	6.8000	1.5177	915.3000	1.0184

Table 4.18: GPMCC method: Initial window 0.1, window acceleration 0.02

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$		σ_{GMSE}	
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$		$\sigma_{R^2_G}$	
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$		$\sigma_{R^2_{aG}}$	
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$		$\frac{\overline{TMSE}}{\overline{GMSE}}$	
Abalone	4.5585	0.0377	5.1005		0.3779	
	0.5639	0.0036	0.5688		0.0320	
	0.5594	0.0029	0.5310		0.0309	
	3.9333	1.1427	1733.3700		0.8937	
Elevators	0.0000	0.0000	0.0000	††	0.0000	
	0.8841	0.0087	0.8807	†	0.0106	
	0.8832	0.0085	0.8731	†	0.0094	
	7.4000	2.3134	2400.4000		1.0000	††
Federal	0.0441	0.0029	0.0811		0.1728	
Reserve	0.9962	0.0003	0.9928		0.0153	
Economic	0.9960	0.0002	0.9879		0.0299	
Data	5.4667	1.5477	933.7000		0.5437	
Function	0.0999	0.0030	0.0900		0.0036	
Example	1.0000	0.0000	1.0000		0.0000	
	1.0000	0.0000	1.0000		0.0000	
	8.1333	1.2521	1372.0700		1.1108	
House-16H	1698038000.0000	73046600.0000	1818504000.0000		229100000.0000	
	0.3894	0.0263	0.3605		0.0796	
	0.3866	0.0259	0.3375		0.0762	
	10.1333	3.0027	3985.0300		0.9338	
Lena	119.2130	3.8666	123.7320	†	6.0586	
Image	0.9412	0.0019	0.9394	†	0.0030	
	0.9409	0.0019	0.9365	†	0.0031	
	14.3333	3.6515	2153.5300		0.9635	†
Mono	300.2740	5.1783	301.5820		6.6504	
Sample	0.6885	0.0054	0.6814		0.0070	
	0.6880	0.0053	0.6774		0.0069	
	10.1333	2.3887	3157.3700		0.9957	
Stereo	201.4200	2.3580	201.7480		2.9485	
Sample	0.8233	0.0021	0.8179		0.0027	
	0.8231	0.0021	0.8161		0.0027	
	5.4667	1.5477	5540.6700		0.9984	
Time-	1.1575	1.5759	1.0481		1.3298	
series	0.9908	0.0126	0.9897		0.0130	
	0.9900	0.0138	0.9689		0.0463	
	6.8667	1.4794	956.7670		1.1044	

Table 4.19: GPMCC method: Initial window 0.1, window acceleration 0.04

Dataset	$\overline{TMSE}$ $\overline{R^2_T}$ $\overline{R^2_{aT}}$ $\overline{Nodes}$	σ_{TMSE} $\sigma_{R^2_T}$ $\sigma_{R^2_{aT}}$ σ_{Nodes}	$\overline{GMSE}$ $\overline{R^2_G}$ $\overline{R^2_{aG}}$ $\bar{t}$	σ_{GMSE} $\sigma_{R^2_G}$ $\sigma_{R^2_{aG}}$ $\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5607	0.0280	5.079160 †	0.3547
	0.5637	0.0027	0.570595 †	0.0300
	0.5591	0.0022	0.531629 †	0.0305
	4.2667	1.2299	1737.4000	0.8979 †
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8834	0.0074	0.8791	0.0085
	0.8825	0.0073	0.8712	0.0078
	6.8667	1.8144	2456.1300	1.0000
Federal	0.0428 †	0.0023	0.0504	0.0078
Reserve	0.9963 †	0.0002	0.9955	0.0007
Economic	0.9961 †	0.0002	0.9930	0.0015
Data	5.6667	1.6046	947.6000	0.8486
Function	0.0995 †	0.0028	0.0901	0.0036
Example	1.0000 †°	0.0000	1.0000	0.0000
	1.0000 †°	0.0000	1.0000	0.0000
	8.1333	1.3578	1358.8300	1.1045
House-16H	1675820000.0000 †	71386200.0000	34665600000.0000	179898600000.0000
	0.3974 †	0.0257	0.3566	0.1065
	0.3944 †	0.0254	0.3346	0.1005
	11.1333	2.7258	3942.6000	0.0483
Lena	119.6078	3.1367	128.6118	10.1046
Image	0.9410	0.0015	0.9370	0.0050
	0.9406	0.0015	0.9340	0.0051
	14.6000	3.8739	2175.1700	0.9300
Mono	300.3680	5.5149	301.5000	6.9080
Sample	0.6884	0.0057	0.6815	0.0073
	0.6879	0.0057	0.6777	0.0073
	9.7333	2.2581	3296.1000	0.9962
Stereo	201.5320	2.8834	201.7980	3.3606
Sample	0.8232	0.0025	0.8178	0.0030
	0.8230	0.0025	0.8160	0.0030
	5.4000	2.3723	5862.5000	0.9987
Time-	0.8319	1.2699	0.8069	0.9965
series	0.9934	0.0101	0.9921	0.0098
	0.9928	0.0111	0.9447	0.1809
	7.2667	1.6386	943.7000	1.0310

Table 4.20: GPMCC method: Initial window 1, window acceleration 0

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{\overline{GMSE}}$
Abalone	4.5575	0.0314	5.1746	0.3875
	0.5640	0.0030	0.5625	0.0328
	0.5596	0.0028	0.5248	0.0335
	4.2000	1.1265	1779.1700	0.8807
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8827	0.0077	0.8791	0.0105
	0.8818	0.0075	0.8712	0.0097
	7.1333	2.4598	2490.6300	1.0000
Federal	0.0446	0.0019	0.0494	0.0063
Reserve	0.9961	0.0002	0.9956	0.0006
Economic	0.9959	0.0002	0.9933	0.0011
Data	5.8667	1.1366	961.1330	0.9022
Function	0.0999	0.0023	0.0907	0.0034
Example	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.6000	1.0700	1471.8700	1.1014
House-16H	1696134000.0000	63944000.0000	2047700000.0000	1140178000.0000
	0.3901	0.0230	0.3426	0.0994
	0.3876	0.0226	0.3230	0.0944
	9.8000	2.6050	4036.0700	0.8283
Lena	120.1524	3.7302	132.8084	33.1722
Image	0.9407	0.0018	0.9349	0.0163
	0.9404	0.0018	0.9316	0.0176
	14.6667	3.9683	2269.8000	0.9047
Mono	299.0820 †	4.0010	300.1520 †	5.1448
Sample	0.6897 †	0.0042	0.6829 †	0.0054
	0.6892 †	0.0041	0.6788 †	0.0055
	11.0667	3.2156	3149.5700	0.9964
Stereo	201.8060	3.1868	202.1960	3.7328
Sample	0.8229	0.0028	0.8175	0.0034
	0.8227	0.0028	0.8157	0.0034
	5.2667	1.7991	6296.5000	0.9981
Time-	0.6623	1.2698	0.5565	0.8542
series	0.9947	0.0101	0.9946	0.0084
	0.9944	0.0108	0.9879	0.0197
	6.8000	1.3235	930.4670	1.1902

tion"). A larger injection rate intuitively leads to a longer optimisation time, because more patterns are iterated over to calculate fitness values *etc.* Similarly, a larger initial window size leads to a longer optimisation time. A window acceleration of less than 0.005 has the consequence that not all patterns are presented to the GPMCC method before the maximum number of generations are reached.

Tables 4.12 - 4.20 show the effect of various initial window sizes and window acceleration rates on the outcomes of the GPMCC method. As expected, a larger initial window and window acceleration leads to an increase in the average simulation completion time $\bar{t}$. However, there is no general relationship between the two window parameters and any of the other outcomes (MSE, nodes, R^2, R_a^2 *etc.*).

The windowing parameters are problem specific, but what is interesting to note is that the continual presentation of all training patterns (shown by table 4.20) was not the strategy that resulted any optimal outcomes (with the exception of the *mono* dataset). Therefore, the initial window parameter should be chosen as 0.05 and the window acceleration parameter should be chosen as 0.005, because these parameters ensure that all training patterns are presented in a timely manner (this would not be the case if the initial window parameter was less than 0.05). Also, these parameter choices result in the smallest training times.

Fragment pool initialisation

The fragment pool initialisation is controlled by two parameters: the number of initial clusters ("DecisionPoolNoClustersStart" in table 4.2) and the split factor ("DecisionPoolNoClusters-Division"). The initial size of the fragment pool is determined by the two parameters, *e.g.* if the initial number of clusters is 30 and the split factor is 2, the initial size of the fragment pool is $30+15+7+3=55$. As was discussed in section 4.3.3, the parameters also control the number of piecewise approximations fitted over the training patterns. These piecewise approximations are then used as terminal nodes for the GPMCC method.

Tables 4.21 - 4.28 show the effect of various initial clusters numbers and window acceleration rates on the outcomes of the GPMCC method. There is no general relationship between the two fragment pool initialisation parameters and any of the outcomes (MSE, nodes, R^2, R_a^2 *etc.*). The initialisation parameters are thus fairly problem specific. Therefore, the two parameters can be chosen arbitrarily. For the remainder of this thesis the number of initial clusters

Table 4.21: GPMCC method: Initial clusters 100, split factor 2 (196 initial fragments)

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\sigma_{TMSE}}{\overline{TMSE}}$
Abalone	4.5595	0.0348	5.1538 †	0.5150
	0.5638	0.0033	0.5643 †	0.0435
	0.5589	0.0027	0.5215	0.0476
	4.5333	1.6344	1556.9700	0.8847 †
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8818	0.0069	0.8763	0.0078
	0.8809	0.0067	0.8684	0.0071
	6.5333 †	2.1453	2275.1300	1.0000
Federal Reserve Economic Data	0.0447	0.0030	0.0472 †	0.0054
	0.9961	0.0003	0.9958 †	0.0005
	0.9959	0.0003	0.9939 †	0.0011
	5.0000	1.3896	879.2000	0.9467 †
Function Example	0.1005	0.0025	0.0920	0.0028
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.4000	0.8137	1078.2300	1.0931
House-16H	1719294000.0000	59086400.0000	1806200000.0000	76553400.0000
	0.3818	0.0212	0.3646	0.0269
	0.3792	0.0209	0.3432	0.0257
	9.6000	3.0240	3538.8000	0.9519
Lena Image	121.3478	3.7463	126.8866	6.0072
	0.9401	0.0018	0.9378	0.0029
	0.9398	0.0018	0.9353	0.0031
	12.0667 †	4.4793	2212.6300	0.9563
Mono Sample	304.8100	7.1109	307.0360	9.0024
	0.6838	0.0074	0.6756	0.0095
	0.6833	0.0073	0.6716	0.0093
	10.2000	2.9989	3060.1700	0.9928
Stereo Sample	202.5780	2.9978	203.1580	3.4670
	0.8223	0.0026	0.8166	0.0031
	0.8220	0.0026	0.8149	0.0031
	4.8667	1.8144	5669.7000	0.98
Time-series	1.1491	1.6181	1.0057	1.3210
	0.9908	0.0129	0.9902	0.0130
	0.9900	0.0143	0.9142	0.2508
	6.5333	1.4559	906.6670	1.1425

Table 4.22: GPMCC method: Initial clusters 100, split factor 3 (143 initial fragments)

Dataset	$\overline{TMSE}$ $\overline{R^2_T}$ $\overline{R^2_{aT}}$ $\overline{Nodes}$	σ_{TMSE} $\sigma_{R^2_T}$ $\sigma_{R^2_{aT}}$ σ_{Nodes}	$\overline{GMSE}$ $\overline{R^2_G}$ $\overline{R^2_{aG}}$ $\bar{t}$	σ_{GMSE} $\sigma_{R^2_G}$ $\sigma_{R^2_{aG}}$ $\frac{\overline{TMSE}}{\overline{TMSE}}$
Abalone	4.5804	0.0278	5.2324	0.4345
	0.5618	0.0027	0.5576	0.0367
	0.5574	0.0025	0.5186	0.0386
	3.8667	1.0080	1531.5000 †	0.8754
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8820	0.0067	0.8783	0.0086
	0.8811	0.0066	0.8703	0.0082
	6.8000	1.8458	2218.2700	1.0000
Federal Reserve Economic Data	0.0446	0.0030	0.0490	0.0068
	0.9961	0.0003	0.9957	0.0006
	0.9959	0.0003	0.9934	0.0014
	5.6000	1.4994	816.4330	0.9102
Function Example	0.1009	0.0027	0.0913	0.0039
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.8000	1.3493	1051.6000	1.1057
House-16H	1726956000.0000	58523200.0000	1921038000.0000	370382000.0000
	0.3790	0.0210	0.3344	0.0831
	0.3766	0.0207	0.3149	0.0822
	9.2667	3.3930	3597.3300	0.8990
Lena Image	121.3546	3.7759	130.8084	23.9748
	0.9401	0.0019	0.9359	0.0117
	0.9398	0.0018	0.9332	0.0124
	12.6667	3.0663	2083.3000	0.9277
Mono Sample	306.3280	7.4583	308.5400	9.5733
	0.6822	0.0077	0.6740	0.0101
	0.6818	0.0077	0.6706	0.0099
	8.5333 †	2.5560	3178.8700	0.9928
Stereo Sample	202.4640	2.3244	202.8180	3.0523
	0.8224	0.0020	0.8169	0.0028
	0.8222	0.0021	0.8154	0.0029
	4.4000	1.1919	5500.2000	0.9983
Time-series	0.8853	0.9890	0.7180	0.6632
	0.9929	0.0079	0.9930	0.0065
	0.9925	0.0084	0.9514	0.1807
	6.8667	1.2794	847.0670	1.2329 †

Table 4.23: GPMCC method: Initial clusters 30, split factor 2 (55 initial fragments)

Dataset	$\overline{TMSE}$ $\overline{R^2_T}$ $\overline{R^2_{aT}}$ $\overline{Nodes}$	σ_{TMSE} $\sigma_{R^2_T}$ $\sigma_{R^2_{aT}}$ σ_{Nodes}	$\overline{GMSE}$ $\overline{R^2_G}$ $\overline{R^2_{aG}}$ $\bar{t}$	σ_{GMSE} $\sigma_{R^2_G}$ $\sigma_{R^2_{aG}}$ $\frac{\overline{TMSE}}{\overline{TMSE}}$
Abalone	4.5750	0.0295	17.6606	67.9024
	0.5624	0.0028	0.5365	0.1071
	0.5576	0.0026	0.4963	0.1018
	4.1333	1.1366	1629.5000	0.2591
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8821	0.0058	0.8783	0.0079
	0.8811	0.0057	0.8702	0.0081
	7.0000	2.1656	2090.2300	1.0000
Federal Reserve Economic Data	0.0438 †	0.0028	0.0485	0.0073
	0.9962 †	0.0002	0.9957	0.0006
	0.9960 †	0.0002	0.9935	0.0016
	5.2000	1.5177	842.9670	0.9037
Function Example	0.0991 †	0.0040	0.0906	0.0045
	1.0000 †°	0.0000	1.0000	0.0000
	1.0000 †°	0.0000	1.0000	0.0000
	7.6667	1.0933	1047.4000	1.0931
House-16H	1715464000.0000	58716400.0000	1799968000.0000	152635400.0000
	0.3832	0.0211	0.3668	0.0537
	0.3806	0.0209	0.3457	0.0538
	8.8000	2.7966	3466.3000	0.9531
Lena Image	120.5956	3.2237	126.3778	4.7560
	0.9405	0.0016	0.9381	0.0023
	0.9402	0.0016	0.9352	0.0026
	13.5333	3.9977	2006.9700	0.9542
Mono Sample	301.5000	5.6420	302.7800	6.9198
	0.6872	0.0059	0.6801	0.0073
	0.6868	0.0058	0.6766	0.0073
	9.0667	1.5298	2751.3700	0.9958
Stereo Sample	201.4420	2.1989	201.5840	2.5408
	0.8232	0.0019	0.8180	0.0023
	0.8231	0.0019	0.8164	0.0023
	4.6667	1.4933	4860.7700	0.9993
Time-series	1.1043	1.4290	1.0168	1.2497
	0.9912	0.0114	0.9901	0.0122
	0.9906	0.0122	0.9749	0.0405
	6.6667	1.4933	807.4330	1.0860

Table 4.24: GPMCC method: Initial clusters 30, split factor 3 (43 initial fragments)

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{TMSE}$
Abalone	4.5708	0.0315	5.3082	0.5044
	0.5628	0.0030	0.5512	0.0426
	0.5586	0.0027	0.5147	0.0436
	3.6667 †	1.0933	1654.1300	0.8611
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8831	0.0049	0.8789	0.0055
	0.8822	0.0048	0.8705	0.0058
	7.2667	2.3332	2065.1300	1.0000
Federal	0.0441	0.0024	0.0541	0.0109
Reserve	0.9961	0.0002	0.9952	0.0010
Economic	0.9960	0.0002	0.9929	0.0017
Data	5.5333	1.3830	764.8000	0.8151
Function	0.0999	0.0033	0.0901 †	0.0038
Example	1.0000	0.0000	1.0000 †°	0.0000
	1.0000	0.0000	1.0000 †°	0.0000
	7.6667	1.5162	1029.6700	1.1085 †
House-16H	1701230000.0000	72622800.0000	1948646000.0000	735114000.0000
	0.3883	0.0261	0.3484	0.0893
	0.3858	0.0258	0.3278	0.0873
	8.8000	2.4269	3403.9300	0.8730
Lena	120.4616	3.0208	125.6458	5.0534
Image	0.9406	0.0015	0.9384	0.0025
	0.9403	0.0015	0.9358	0.0024
	12.9333	3.8768	1928.7700 †	0.9587
Mono	301.0580	5.6103	302.0440	6.8308
Sample	0.6876	0.0058	0.6809	0.0072
	0.6872	0.0058	0.6768	0.0070
	10.3333	2.6436	2722.6000 †	0.9967
Stereo	201.9240	2.3849	202.2740	2.7546
Sample	0.8228	0.0021	0.8174	0.0025
	0.8226	0.0021	0.8158	0.0025
	4.5333	1.6344	4850.4000	0.9983
Time-	1.0636	1.4651	0.9553	1.3346
series	0.9915	0.0117	0.9907	0.0131
	0.9909	0.0124	0.9779	0.0316
	7.1333	2.0297	810.4670	1.1133

Table 4.25: GPMCC method: Initial clusters 20, split factor 2 (37 initial fragments)

Dataset	$\overline{TMSE}$ $\overline{R^2_T}$ $\overline{R^2_{aT}}$ $\overline{Nodes}$	σ_{TMSE} $\sigma_{R^2_T}$ $\sigma_{R^2_{aT}}$ σ_{Nodes}	$\overline{GMSE}$ $\overline{R^2_G}$ $\overline{R^2_{aG}}$ $\bar{t}$	σ_{GMSE} $\sigma_{R^2_G}$ $\sigma_{R^2_{aG}}$ $\frac{\overline{TMSE}}{\overline{TMSE}}$
Abalone	4.5531 † 0.5645 † 0.5598 † 4.2000	0.0359 0.0034 0.0032 1.4480	5.3183 0.5504 0.5081 1641.3000	0.6657 0.0563 0.0649 0.8561
Elevators	0.0000 0.8830 0.8820 7.1333	0.0000 0.0071 0.0071 1.8889	0.0000 †† 0.8798 † 0.8714 † 2032.7700	0.0000 0.0088 0.0088 1.0000 ††
Federal Reserve Economic Data	0.0445 0.9961 0.9959 5.6000	0.0024 0.0002 0.0002 1.6733	0.0502 0.9956 0.9931 781.1000	0.0074 0.0007 0.0018 0.8870
Function Example	0.0997 1.0000 1.0000 7.4667 †	0.0031 0.0000 0.0000 1.0080	0.0914 1.0000 1.0000 996.4330	0.0046 0.0000 0.0000 1.0915
House-16H	1697054000.0000 0.3898 0.3871 9.0667	72796600.0000 0.0262 0.0259 2.0667	1785874000.0000 0.3718 0.3492 3447.9000	123829600.0000 0.0436 0.0435 0.9503
Lena Image	120.1176 0.9407 0.9404 13.8000	4.2472 0.0021 0.0020 4.1223	123.9618 † 0.9393 † 0.9364 † 1945.0700	7.9181 0.0039 0.0037 0.9690 †
Mono Sample	300.0740 † 0.6887 † 0.6882 † 9.2667	5.3272 0.0055 0.0055 2.0160	301.0620 † 0.6819 † 0.6783 † 2756.1300	6.6463 0.0070 0.0071 0.9967
Stereo Sample	201.0540 0.8236 0.8234 4.8000	1.9399 0.0017 0.0017 1.6060	201.2580 0.8183 0.8167 4654.4000 †	2.3755 0.0021 0.0022 0.9990
Time-series	1.3801 0.9890 0.9883 6.4000	1.6197 0.0129 0.0138 1.6733	1.1452 0.9888 0.9739 825.5330	1.2081 0.0118 0.0296 1.2051

Table 4.26: GPMCC method: Initial clusters 20, split factor 3 (28 initial fragments)

Dataset	$\overline{TMSE}$ $\overline{R^2_T}$ $\overline{R^2_{aT}}$ $\overline{Nodes}$	σ_{TMSE} $\sigma_{R^2_T}$ $\sigma_{R^2_{aT}}$ σ_{Nodes}	$\overline{GMSE}$ $\overline{R^2_G}$ $\overline{R^2_{aG}}$ $\bar{t}$	σ_{GMSE} $\sigma_{R^2_G}$ $\sigma_{R^2_{aG}}$ $\frac{\overline{TMSE}}{\overline{TMSE}}$
Abalone	4.5626	0.0367	5.4460	1.4684
	0.5636	0.0035	0.5426	0.1087
	0.5587	0.0029	0.5016	0.1042
	4.3333	1.4223	1610.3300	0.8378
Elevators	0.0000 ††	0.0000	0.0000	0.0000
	0.8832 †	0.0076	0.8792	0.0091
	0.8823 †	0.0075	0.8713	0.0086
	7.0000	2.0342	2029.2000	1.0000
Federal Reserve Economic Data	0.0443	0.0024	0.0510	0.0093
	0.9961	0.0002	0.9955	0.0008
	0.9960	0.0002	0.9933	0.0014
	5.2667	1.1427	772.4670	0.8681
Function Example	0.1006	0.0027	0.0911	0.0034
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.8667	1.3578	1041.3700	1.1033
House-16H	1689180000.0000	76237000.0000	1763444000.0000	91387200.0000
	0.3926	0.0274	0.3797	0.0321
	0.3898	0.0271	0.3565	0.0311
	9.7333	2.8519	3276.7700 †	0.9579
Lena Image	119.8186	3.2694	128.9602	12.2305
	0.9409	0.0016	0.9368	0.0060
	0.9406	0.0016	0.9339	0.0061
	12.6667	3.6797	2015.2000	0.9291
Mono Sample	300.6600	6.1792	301.5380	7.8649
	0.6881	0.0064	0.6814	0.0083
	0.6876	0.0064	0.6778	0.0082
	9.8667	2.3302	2787.3300	0.9971 †
Stereo Sample	201.7160	2.4743	202.0160	3.3177
	0.8230	0.0022	0.8176	0.0030
	0.8228	0.0022	0.8161	0.0031
	4.8000	1.3235	4686.8700	0.9985
Time-series	1.4982	1.7086	1.2820	1.4768
	0.9881	0.0136	0.9875	0.0145
	0.9872	0.0145	0.9581	0.0639
	6.2667 †	1.7006	845.2000	1.1686

Table 4.27: GPMCC method: Initial clusters 10, split factor 2 (17 initial fragments)

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{\overline{TMSE}}$
Abalone	4.5563	0.0303	5.1816	0.3516
	0.5642	0.0029	0.5619	0.0297
	0.5596	0.0025	0.5218 †	0.0323
	4.0667	1.2576	1590.4700	0.8793
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8830	0.0062	0.8789	0.0074
	0.8821	0.0060	0.8711	0.0074
	7.0000	2.1656	2030.0300	1.0000
Federal Reserve Economic Data	0.0446	0.0022	0.0478	0.0069
	0.9961	0.0002	0.9958	0.0006
	0.9959	0.0002	0.9937	0.0011
	4.9333 †	1.229900	814.633000	0.9322
Function Example	0.1003	0.0029	0.0920	0.0034
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	7.6667	1.0933	1007.6700	0.8793
House-16H	1681064000.0000 †	71950200.0000	1749322000.0000 †	91357200.0000
	0.3955 †	0.0259	0.3846 †	0.0321
	0.3930 †	0.0256	0.3636 †	0.0309
	8.8000 †	2.9408	3485.8700	0.9610 †
Lena Image	119.7560 †	3.0794	124.9156	5.7129
	0.9409 †	0.0015	0.9388	0.0028
	0.9406 †	0.0015	0.9361	0.0027
	12.8667	3.6741	1972.9000	0.9587
Mono Sample	301.5620	6.5922	302.7680	8.1946
	0.6871	0.0068	0.6801	0.0087
	0.6867	0.0068	0.6765	0.0084
	9.1333	2.2854	2825.9300	0.9960
Stereo Sample	201.7360	2.7894	202.0880	3.4211
	0.8230	0.0024	0.8176	0.0031
	0.8228	0.0024	0.8160	0.0032
	4.5333 †	1.4559	4715.5700	0.9983
Time-series	0.7055 †	1.0167	0.6232 †	0.8189
	0.9944 †	0.0081	0.9939 †	0.0080
	0.9939 †	0.0087	0.9749 †	0.0524
	7.4667	1.2521	750.5000 †	1.1321

Table 4.28: GPMCC method: Initial clusters 10, split factor 3 (13 initial fragments)

Dataset	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
	$\overline{R^2_T}$	$\sigma_{R^2_T}$	$\overline{R^2_G}$	$\sigma_{R^2_G}$
	$\overline{R^2_{aT}}$	$\sigma_{R^2_{aT}}$	$\overline{R^2_{aG}}$	$\sigma_{R^2_{aG}}$
	$\overline{Nodes}$	σ_{Nodes}	$\bar{t}$	$\frac{\overline{TMSE}}{\overline{TMSE}}$
Abalone	4.5692	0.0342	5.2241	0.4925
	0.5629	0.0033	0.5583	0.0416
	0.5585	0.0030	0.5204	0.0447
	3.7333	1.1121	1617.7000	0.8746
Elevators	0.0000	0.0000	0.0000	0.0000
	0.8819	0.0075	0.8783	0.0090
	0.8810	0.0073	0.8701	0.0085
	7.1333	2.5695	1972.7000 †	1.0000
Federal Reserve Economic Data	0.0442	0.0024	0.0488	0.0060
	0.9961	0.0002	0.9957	0.0005
	0.9960	0.0002	0.9933	0.0015
	5.6000	1.5888	751.1000 †	0.9043
Function Example	0.0999	0.0031	0.0907	0.0042
	1.0000	0.0000	1.0000	0.0000
	1.0000	0.0000	1.0000	0.0000
	8.0667	1.6386	948.8670 †	1.1013
House-16H	1708460000.0000	77513400.0000	1779002000.0000	128243800.0000
	0.3857	0.0279	0.3742	0.0451
	0.3830	0.0276	0.3521	0.0444
	9.6000	2.5271	3325.8700	0.9603
Lena Image	120.2276	3.6120	125.0488	5.6104
	0.9407	0.0018	0.9387	0.0027
	0.9404	0.0018	0.9359	0.0027
	13.4667	2.8616	1976.2000	0.9614
Mono Sample	303.2560	6.3768	305.3640	8.0327
	0.6854	0.0066	0.6774	0.0085
	0.6849	0.0066	0.6734	0.0086
	9.4667	2.3887	2876.6000	0.9931
Stereo Sample	200.6680 †	1.7839	200.7440 †	2.1620
	0.8239 †	0.0016	0.8188 †	0.0020
	0.8237 †	0.0015	0.8170 †	0.0019
	4.8000	1.9191	4771.5700	0.9996 †
Time-series	1.0371	1.5863	0.8519	1.2269
	0.9917	0.0127	0.9917	0.0120
	0.9912	0.0136	0.9803	0.0358
	6.6667	1.2954	786.6330	1.2175

is set to 30 and the split factor is set to 2. This should provide enough clusters to cover the turning points of a dataset.

4.4.3 Method Comparison

Although a large number of initial parameters were introduced in the previous section, the GPMCC method appears to be fairly robust in that different values for parameters do not have a significant effect on accuracy. This section compares the GPMCC method to two other methods discussed earlier in this thesis.

The first comparison method is Setiono's NeuroLinear method from section 4.2.1. Setiono presents a table of results, obtained by running NeuroLinear on 5 databases from the UCI machine learning repository [91]. The databases used by Setiono are the Abalone, Auto-Mpg, Housing, Machine and Servo databases discussed in section 4.4.1. Setiono performed one 10-fold cross validation evaluation on each of the previously mentioned datasets. The predictive accuracy of NeuroLinear was tested in terms of the generalisation *mean absolute error*:

$$E_{MA} = \frac{\sum_{i=1}^{|P|} |y_i - y_i^{\bullet}|}{|P|}$$

Setiono also provided the average number of rules generated for each dataset.

The second comparison method is a commercial version of the M5 algorithm (successor to C4.5) called Cubist [82]. Cubist internally utilises model trees with linear regression models. Cubists presents these model trees in the form of a production system. Both Cubist and the GPMCC method were used to perform 10 10-fold cross validation evaluations (equivalent to 100 simulation runs) on the datasets mentioned previously, in order to determine the generalisation mean absolute error. The average number of generated rules, rule conditions (the length of the path from the root to the terminal node) and rule terms (the number of terms in the model) were also obtained for each dataset. Table 4.29 shows the initialisation parameters for the GPMCC method as determined by the findings of the previous section. For all datasets the maximum polynomial order was set to 5, except for the house-16H dataset which was set to 10.

Table 4.30 shows the results for Cubist, the GPMCC method and NeuroLinear for the 5 databases mentioned above. In all the cases the GPMCC method was the least accurate of the

Table 4.29: Changes in GPMCC initialisation parameters from table 4.2

Parameter	Value
DecisionReoptimizeVsSelectLeaf	0.05
DecisionPoolNoClustersStart	30
DecisionPoolNoClustersDivision	2
DecisionPoolFragmentLifeTime	50
DecisionInitialPercentageSampleSize	0.05
DecisionSampleAcceleration	0.005
DecisionCrossoverRate	0.1
CrossValidation	1

Table 4.30: Comparison of Cubist, GPMCC and NeuroLinear ($\overline{GMAE}$ is the average generalisation mean absolute error, σ_{GMAE} is the standard deviation for the generalisation mean absolute error, $\overline{rules}$ represents the average number of rules, σ_{rules} represents the standard deviation for the number of rules, $\overline{conds}$ represents the average number of rule conditions, σ_{conds} is the standard deviation for the number of rule conditions, $\overline{terms}$ represents the average number of rules per term and σ_{terms} is the standard deviation for the number of rules per term)

Dataset	Method	$\overline{GMAE}$	σ_{GMAE}	$\overline{rules}$	σ_{rules}	$\overline{conds}$	σ_{conds}	$\overline{terms}$	σ_{terms}
Abalone	Cubist	1.4950	0.0609	12.6500	4.1056	2.6545	0.5700	4.7560	0.4978
	GPMCC	1.6051	0.0156	2.5400	0.6264	1.3407	0.3688	8.3213	1.0634
	NL	1.5700	0.0600	4.1000	1.4500	n/a	n/a	n/a	n/a
Auto-Mpg	Cubist	1.8676	0.2536	5.1000	1.3890	2.0816	0.4225	3.0013	0.4972
	GPMCC	2.1977	0.3703	2.1800	0.6724	1.0765	0.4985	7.2195	1.6180
	NL	1.9600	0.3200	7.5000	5.0800	n/a	n/a	n/a	n/a
Housing	Cubist	1.7471	0.2633	12.6700	3.3727	3.2125	0.4542	5.2164	0.6068
	GPMCC	2.8458	0.5217	2.8200	0.7962	1.5103	0.4534	7.6613	1.4220
	NL	2.5300	0.4600	25.3000	17.1300	n/a	n/a	n/a	n/a
Machine	Cubist	26.9280	7.5873	4.8800	1.4162	1.9367	0.4106	4.6818	0.7785
	GPMCC	34.3228	17.0849	3.4600	0.9773	1.8770	0.4983	3.4095	0.9773
	NL	20.9900	11.3800	3.0000	3.0000	n/a	n/a	n/a	n/a
Servo	Cubist	0.3077	0.1252	9.6100	1.9638	2.7298	0.3027	2.1033	0.2890
	GPMCC	0.4496	0.1755	5.1500	1.9456	2.6247	0.8876	2.7935	0.8482
	NL	0.3400	0.0800	4.7000	2.3100	n/a	n/a	n/a	n/a

three methods in terms of generalisation accuracy (but not significantly). Cubist, on the other hand, was the most accurate of the three methods. However, the number of rules generated by the GPMCC method was significantly less than that of the other methods, with the exception of the Servo and Machine datasets. Also, the total complexity of the GPMCC method, in terms of the average number of rules, the average number of rule conditions and the average number of rule terms was significantly less than that of Cubist.

The GPMCC method and Cubist were also compared on the remaining datasets not shown in table 4.30. Once again, 10 10-fold cross validation evaluations were performed in order to obtain the generalisation mean absolute error. Additionally, the average number of generated rules, rule conditions and rule terms were obtained for each database. The GPMCC method was once again initialised using table 4.29.

Table 4.31: Comparison of Cubist and GPMCC

Dataset	Method	$\overline{GMAE}$	σ_{GMAE}	$\overline{rules}$	σ_{rules}	$\overline{conds}$	σ_{conds}	$\overline{terms}$	σ_{terms}
Elevators	Cubist	0.0019	0.0001	18.0400	2.3047	2.9669	0.3206	2.9493	0.3229
	GPMCC	0.0018	0.0000	3.3200	0.9522	1.7929	0.5161	8.8217	0.8498
House-16H	Cubist	16355.2840	375.0254	35.7100	4.7637	4.4092	0.4649	4.1570	0.4291
	GPMCC	24269.8000	3889.8900	5.1300	1.2032	2.6361	0.5451	7.2623	1.5700
Federal	Cubist	0.0999	0.0137	17.0300	4.0613	2.5765	0.3249	4.8203	0.4984
Reserve ED	GPMCC	0.1514	0.0366	2.8700	0.6765	1.5553	0.4031	7.6408	1.1640
Function	Cubist	0.9165	1.4499	41.4600	5.3756	2.8032	0.2843	1.1384	0.1221
Example	GPMCC	1.3713	1.8050	4.1700	0.5695	2.0916	0.2158	2.1400	0.1654
Lena	Cubist	5.0160	0.2102	32.8000	4.5969	4.3619	0.4775	3.9577	0.3999
Image	GPMCC	6.4107	0.2466	6.5500	1.6229	3.0743	0.5332	7.4953	1.0319
Mono	Cubist	13.3550	0.1749	35.2700	4.5480	3.6664	0.3942	4.4224	0.4533
Sample	GPMCC	13.7162	0.1762	4.5300	1.1845	2.4217	0.5613	4.7366	0.4293
Stereo	Cubist	11.2470	0.1507	11.5500	2.8298	2.9741	0.4580	4.9500	0.5000
Sample	GPMCC	11.2498	0.1923	2.6100	0.7092	1.3883	0.4392	7.8308	0.6162
Time-	Cubist	0.7664	0.1853	30.2400	3.8379	3.5024	0.3778	3.1646	0.3315
series	GPMCC	0.6442	0.3718	3.6700	0.8996	2.0062	0.4271	5.0386	1.6865

Table 4.31 shows the results for Cubist and the GPMCC method for the remaining databases not used in table 4.30. In all cases the GPMCC method outperformed Cubist in terms of the average number of rules generated and the total complexity. In fact, the average number of rules generated by the GPMCC method and the total complexity of the GPMCC method were significantly less than that of Cubist. Additionally, the GPMCC method even managed to outperform Cubist in terms of the generalisation mean absolute error on some of the datasets, *i.e.* Elevators and Time-series. However, there is no statistically significant difference in accuracy.

4.4.4 Rule quality

This section discusses the quality of the rules inferred by the GPMCC method for each of the datasets in section 4.4.1. Each model tree represents the best outcome of 10 10-fold cross validation evaluations in terms of the mean squared error on the generalisation set. The GPMCC method was initialised using table 4.29. Values between parenthesis show how many patterns are covered by the antecedent of a rule. Values between angle brackets indicate the mean squared error of the rule on patterns covered by the antecedent of that rule. For both types of parenthesis, the first value between parenthesis represents the outcome for the training set and the second value represents the outcome for the validation set.

- The best model tree describing the **Abalone** dataset is as follows:

```
if (Sex == "F") {
  if (Viscera > 0.545792) {
    Rings = 13.6301*pow(Shell,1)
           -16.1805*pow(Viscera,1)
           -26.6776*pow(Shucked,1)
           +13.0827*pow(Whole,1)
           +8.99643;
    //(1, 0) <48.201, 0>
  } else {
    Rings =-44.7887*pow(Shell,1)*pow(Length,1)
           +42.2753*pow(Shell,1)
           +132.724*pow(Viscera,3)*pow(Diameter,2)
           -21.1746*pow(Viscera,1)
           +22.7078*pow(Shucked,2)
           -45.8945*pow(Shucked,1)
           -6.61542*pow(Whole,2)
           +28.4355*pow(Whole,1)
           +0.782952*pow(Height,1)
           +4.46808;
    //(1027, 130) <6.01941, 5.99693>
  }
} else {
  Rings = 13.1915*pow(Shell,1)
         +17.8778*pow(Viscera,1)*pow(Whole,1)*pow(Diameter,1)
         -40.99*pow(Viscera,1)*pow(Length,1)
         +59.2038*pow(Shucked,2)
```

```
          -35.4237*pow(Shucked,1)*pow(Whole,1)
          -42.489*pow(Shucked,1)
          +0.10544*pow(Whole,4)
          +27.2256*pow(Whole,1)
          +4.30859*pow(Diameter,1)
          +3.96395;
  //(2313, 288) <3.83083, 3.1415>
}
TMSE: 4.51687
VMSE: 4.02955
GMSE: 4.904
```

The largest order used by the models of the model tree is 5. Also, the first rule represents an outlier. Obviously this outlier skewed the coefficients of the GPMCC method so drastically that the GPMCC method had no choice but to isolate it. This indicates that this training pattern should be removed from the dataset.

- The best model tree describing the **Auto-mpg** dataset is as follows:

```
if (displacement > 97.2829) {
  mpg = -0.061885*pow(model_year,1)*pow(cylinders,1)
        +0.933105*pow(model_year,1)
        +0.00151765*pow(weight,1)*pow(cylinders,1)
        -0.0147998*pow(weight,1)
        -0.037649*pow(horsepower,1);
  //(251, 30) <7.37801, 7.72467>
} else {
  mpg = -1.18039*pow(origin,1)
        +0.0479332*pow(model_year,2)
        -0.0172612*pow(model_year,1)*pow(cylinders,2)
        -5.99653*pow(model_year,1)
        -0.00545806*pow(weight,1)
        -0.0244613*pow(horsepower,1)
        -0.0546243*pow(displacement,1)
        +14.5401*pow(cylinders,1)
        +192.894;
  //(61, 10) <17.2217, 3.97088>
}
TMSE: 9.30258
VMSE: 6.78622
```

```
GMSE: 3.48597
```

Only two non-linear rules where generated. The largest order utilised by the models of the model tree is 3.

- The best model tree describing the **Elevators** dataset is as follows:

```
if (SaTime1 < -0.000562935) {
  if (diffRollRate > -0.011995) {
    Goal = 0.0848297*pow(Sa,1)*pow(diffClb,1)
           -56.4038*pow(Sa,1)
           -9511.27*pow(SaTime4,2)
           -0.00331258*pow(SaTime3,1)*pow(climbRate,1)
           -1.43111*pow(SaTime1,1)
           +0.747046*pow(diffRollRate,1)
           +0.00236211*pow(absRoll,1)
           +0.00547074*pow(p,1)
           +0.0106479;
    //(2925, 372) <7.95712e-06, 5.63356e-06>
  } else {
    Goal = -36.6557*pow(Sa,1)
           -4101.46*pow(SaTime4,2)
           -0.00368597*pow(SaTime3,1)*pow(climbRate,1)
           +0.0014319*pow(SaTime2,1)*pow(Sgz,1)
           +0.463288*pow(diffRollRate,1)
           +0.00154505*pow(absRoll,1)
           +0.00781973*pow(q,1)
           +0.00305086*pow(p,1)
           +0.0123348;
    //(1189, 151) <3.92677e-06, 3.27794e-06>
  }
} else {
  Goal = -25.0523*pow(Sa,1)
         +0.116456*pow(SaTime4,1)*pow(diffClb,1)
         -0.00669548*pow(SaTime3,1)*pow(climbRate,1)
         +0.433775*pow(diffRollRate,1)
         +0.00126904*pow(absRoll,1)
         +0.00372367*pow(p,1)
         +0.016304;
  //(2886, 353) <4.34178e-06, 4.54399e-06>
```

```
}
TMSE: 5.78198e-06
VMSE: 4.78845e-06
GMSE: 4.41977e-06
```

A large number of terms were utilised by the models of the model tree. However, the maximum polynomial order of the models in the model tree is 4.

- The best model tree describing the **Federal Reserve Economic Data** dataset is as follows:

```
if (Y3TCMR > 15.8756) {
  M1CDR = 0.04954*pow(TWEIMC,1)
          +0.00308282*pow(M3TBRAA,2);
  //(3, 1) <0.246856, 0.641734>
} else {
  if (M3TBRSM > 11.1668) {
    M1CDR = -0.132345*pow(TLLACB,1)
            +0.0941262*pow(TCD,1)
            +0.56461*pow(M1MS,1)
            -0.162584*pow(BCACB,1)
            +0.206319*pow(Y3TCMR,1)
            -0.0400925*pow(M3TBRAA,1)
            +0.446434*pow(Y30CMR,1);
    //(82, 8) <0.11477, 0.0222504>
  } else {
    M1CDR = 0.450881*pow(M1MS,1)
            -0.0102812*pow(DDCB,1)
            +0.00126507*pow(CCMS,1)
            +0.0538425*pow(BCACB,1)
            -0.00205841*pow(Y5TCMR,2)
            -0.356311*pow(Y5TCMR,1)
            +0.00877064*pow(Y3TCMR,2)
            +0.00308282*pow(M3TBRAA,2)
            +0.69845*pow(Y30CMR,1)
            -0.11687;
    //(754, 96) <0.0422099, 0.0382938>
  }
}
TMSE: 0.0500334
VMSE: 0.0428185
GMSE: 0.0250136
```

A large number of terms were utilised by the models of the model tree. The maximum polynomial order of the models in the tree is 2.

- The best model tree describing the **Function Example** dataset is as follows:

```
if (type == "A") {
  y = 2.01245*pow(x,1)
      +0.512857;
  //(266, 22) <0.0912585, 0.0873773>
} else {
  if (type == "C") {
    if (x > 0.997578) {
      y = 1.48816*pow(x,2)
          -6.83703*pow(x,1)
          +2.09168;
      //(66, 9) <0.0959942, 0.0494538>
    } else {
      y = 1.01387*pow(x,3)
          +400.504;
      //(204, 31) <0.100298, 0.0883689>
    }
  } else {
    y = -1.9986*pow(x,1)
        +0.512857;
    //(264, 38) <0.0853964, 0.0836496>
  }
}
TMSE: 0.0920197
VMSE: 0.0828551
GMSE: 0.0811045
```

What is interesting to note, is that the model tree is almost identical to the generating function of section 4.4.1.

- The best model tree describing the **House-16H** dataset is as follows:

```
if (P11p4 < 0.00635415) {
  price = -249056*pow(H10p1,1)*pow(P16p2,1)
          -150165*pow(P27p4,3)
          +286593*pow(P16p2,1);
```

```
    //(123, 12) <3.18889e+09, 4.06234e+09>
  } else {
    if (H40p4 < 0.0122007) {
      if (H13p1 < 0.847563) {
        price = 287953*pow(H13p1,2)
                -292744*pow(H13p1,1)
                -1987.53*pow(H8p2,1)
                +45552.3*pow(H2p2,1)
                +318809*pow(P27p4,1)*pow(P16p2,1)
                +130331*pow(P14p9,6)*pow(P6p2,2)*pow(P1,1)
                -1789.72*pow(P11p4,4)
                +12.4096*pow(P1,1)
                +70607.7;
        //(3177, 397) <1.23449e+09, 6.54793e+08>
      } else {
        price = 49642.6*pow(P15p1,1);
        //(57, 6) <1.19254e+09, 5.26761e+08>
      }
    } else {
      if (H10p1 > 0.965729) {
        if (H13p1 < 0.707063) {
          price = 1.42302e+06*pow(H18pA,1)*pow(H10p1,8)
                  -1.50279e+06*pow(H18pA,1)
                  -1.42104e+06*pow(H13p1,4)
                  +1.78124e+06*pow(H13p1,2)*pow(H10p1,3)
                  -960514*pow(H13p1,1)
                  -2.91708e+06*pow(H10p1,1)
                  +491007*pow(P27p4,1)*pow(P16p2,1)
                  -97214.2*pow(P14p9,1)
                  +3.09173e+06;
          //(12264, 1542) <1.58384e+09, 1.18774e+09>
        } else {
          price = 49642.6*pow(P15p1,1);
          //(61, 2) <1.7607e+09, 1.05144e+09>
        }
      } else {
        if (P18p2 < 0.0693344) {
          price = -49761.3*pow(H40p4,1)
                  +741151*pow(H18pA,2)
                  -352497*pow(H18pA,1)
                  +27277.1*pow(H10p1,2)
```

```
                -86146*pow(H2p2,1)
                +2.06379e+06*pow(P27p4,1)*pow(P5p1,1)
                -5.10314e+06*pow(P18p2,1)*pow(P5p1,1)
                +179486*pow(P16p2,2)
                -8558.44;
          //(2543, 320) <2.70133e+09, 2.22773e+09>
        } else {
          price = 49642.6*pow(P15p1,1);
          //(1, 0) <1.82529e+08, 0>
        }
      }
    }
}
TMSE: 1.68899e+09
VMSE: 1.2542e+09
GMSE: 1.29921e+09
```

A large number of non-linear rules were obtained from the dataset. The maximum utilised polynomial order for the rules was 9.

- The best model tree describing the **Housing** dataset is as follows:

```
if (DIS < 1.81274) {
  MEDV = 0.0311044*pow(LSTAT,2)
         -0.0852524*pow(LSTAT,1)*pow(PTRATIO,1)*pow(NOX,1)
         -0.596869*pow(LSTAT,1)
         +0.431583*pow(RAD,1)*pow(CHAS,1)
         -1.35841*pow(DIS,1)
         +0.119659*pow(RM,3)
         -11.7921*pow(RM,1)
         -0.108755*pow(CRIM,1)
         +83.3143;
  //(63, 8) <33.3665, 3.50684>
} else {
  MEDV = -0.767679*pow(LSTAT,1)*pow(NOX,1)
         -0.729239*pow(PTRATIO,1)
         -0.0137382*pow(TAX,1)
         -0.255243*pow(RAD,1)*pow(RM,1)
         +1.71962*pow(RAD,1)
         -0.59144*pow(DIS,1)
```

```
          +2.81649*pow(RM,2)
          -30.0403*pow(RM,1)
          +1.5585*pow(CHAS,1)
          +124.502;
  //(341, 43) <10.7152, 7.98617>
}
TMSE: 14.2475
VMSE: 7.28353
GMSE: 6.29143
```

Only two non-linear rules were obtained. The maximum order of the models of the model tree is 3.

- The best model tree describing the **Lena Image** dataset is as follows:

```
if (blend[7] < 203.273) {
  intensity = -0.0123416*pow(across,1)
              +0.00189951*pow(blend[7],2)
              -0.0477566*pow(blend[6],1)
              +0.326583*pow(blend[5],1)
              +0.694096*pow(blend[4],1)
              -0.00194599*pow(blend[3],2)
              +0.22087*pow(blend[3],1)
              -0.132875*pow(blend[2],1)
              -0.0649604*pow(blend[1],1)
              +1.96046;
  //(12328, 1532) <127.918, 120.595>
} else {
  if (blend[3] > 118.928) {
    if (blend[0] > 49.8796) {
      if (across < 55.2729) {
        intensity = 0.507778*pow(blend[7],1)
                    -0.0444378*pow(blend[6],1)
                    +0.354999*pow(blend[5],1)
                    +0.659356*pow(blend[4],1)
                    -0.303265*pow(blend[3],1)
                    -0.176485*pow(blend[2],1);
        //(38, 6) <351.494, 62.1143>
      } else {
        intensity = 0.0105132*pow(blend[7],2)
```

```
                    -0.0205424*pow(blend[7],1)*pow(blend[4],1)
                    -0.00172814*pow(blend[5],2)
                    +0.00143276*pow(blend[5],1)*pow(blend[1],1)
                    +0.317081*pow(blend[5],1)
                    +5.42638*pow(blend[4],1)
                    -0.543182*pow(blend[1],1)
                    -0.167443*pow(blend[0],1)
                    -391.867;
        //(734, 100) <150.048, 128.225>
      }
    } else {
      intensity = 1.94436*pow(blend[2],1);
      //(2, 0) <1.63576, 0>
    }
  } else {
    intensity = 1.47793*pow(blend[5],1)
                -0.62556*pow(blend[2],1)
                +0.263644*pow(blend[1],1);
    //(4, 1) <0.00493363, 1451.8>
  }
}
TMSE: 129.747
VMSE: 121.659
GMSE: 106.126
```

Essentially, the rules consist of linear blends of the context pixels to obtain the predicted pixel value. One of the rules represents an outlier. Also, the maximum polynomial order of the models is 2.

- The best model tree describing the **Machine** dataset is as follows:

```
if (CACH < 128.391) {
  if (MMIN < 25112.4) {
    PRP = 0.0347948*pow(CHMAX,1)*pow(CACH,1)
          -0.14693*pow(CHMIN,2)
          +0.00117564*pow(CHMIN,1)*pow(MMIN,1)
          +0.00558463*pow(MMAX,1)
          -0.0010641*pow(MMIN,1);
    //(160, 21) <1212.63, 568.998>
  } else {
```

```
    PRP = 31.0917*pow(CHMIN,1)
          +0.00430516*pow(MMIN,1);
    //(1, 0) <127.71, 0>
  }
} else {
  PRP = -2.33197*pow(CHMAX,1)
        +30.9474*pow(CHMIN,1)
        +0.00440527*pow(MMIN,1);
  //(6, 0) <736.413, 0>
}
TMSE: 1189.02
VMSE: 568.998
GMSE: 333.024
```

A small number of linear rules where generated. The largest order utilised by the models of the model tree is 2. Once again one of the rules represents an outlier.

- The best model tree describing the **Mono Sample** dataset is as follows:

```
if (t < 10566.2) {
  if (buf[0] > 130.813) {
    y = 0.869102*pow(buf[2],1)
        -0.398713*pow(buf[1],1)
        +0.436347*pow(buf[0],1)
        +11.8149;
    //(3932, 497) <307.129, 311.468>
  } else {
    y = 0.8834*pow(buf[2],1)
        -0.454296*pow(buf[1],1)
        +0.475617*pow(buf[0],1)
        +12.0682;
    //(4510, 540) <314.362, 334.597>
  }
} else {
  if (t < 20022.7) {
    if (buf[1] < 145.598) {
      y = -0.00243541*pow(buf[2],1)*pow(buf[1],1)
          +1.17106*pow(buf[2],1)
          +0.00179399*pow(buf[1],2)
          -1.04915*pow(buf[1],1)
```

```
            +0.721939*pow(buf[0],1)
            +28.1399;
        //(5460, 692) <332.485, 356.925>
      } else {
        y = 0.00263594*pow(buf[2],2)
            -0.447044*pow(buf[1],1)
            +0.566219*pow(buf[0],1)
            +61.1108;
        //(2109, 235) <286.787, 252.748>
      }
    } else {
      y = 1.08554*pow(buf[2],1)
          -0.646192*pow(buf[1],1)
          +0.366292*pow(buf[0],1)
          +24.683;
      //(9495, 1225) <263.164, 249.476>
    }
}
TMSE: 295.787
VMSE: 297.108
GMSE: 294.327
```

A small number of linear rules where generated. The largest order utilised by the models of the model tree is 2.

- The best model tree describing the **Servo** dataset is as follows:

```
if (motor == "D") {
  class = -0.343752*pow(pgain,1)
          +2.13126;
  //(17, 2) <0.267481, 0.999964>
} else {
  if (motor == "E") {
    if (screw == "A") {
      class = -0.0971804*pow(vgain,1)*pow(pgain,1)
              +0.59491*pow(vgain,1)
              -0.489579*pow(pgain,3)
              +7.57494*pow(pgain,2)
              -38.6539*pow(pgain,1)
              +65.5058;
```

```
      //(6, 1) <1.43629, 0.00659144>
    } else {
      class = -0.343752*pow(pgain,1)
              +2.13126;
      //(19, 2) <0.126267, 0.00564462>
    }
  } else {
    class = -0.122795*pow(vgain,1)*pow(pgain,1)
            +0.730519*pow(vgain,1)
            -0.447423*pow(pgain,3)
            +7.05858*pow(pgain,2)
            -36.4128*pow(pgain,1)
            +61.5672;
    //(91, 12) <0.39332, 0.016951>
  }
}
TMSE: 0.386136
VMSE: 0.13066
GMSE: 0.0315292
```

A small number of rules where generated, however some of the rules are non-linear. The maximum order of the models of the model tree is 3.

- The best model tree describing the **Stereo Sample** dataset is as follows:

```
if (t > 23500.3) {
  y = 0.510853*pow(left[3],1)
      +0.754001*pow(right[2],1)
      -0.0890549*pow(left[2],1)
      -0.677194*pow(right[1],1)
      +0.333603*pow(left[1],1)
      +0.393169*pow(right[0],1)
      -0.180878*pow(left[0],1)
      -5.84898;
  //(6356, 797) <198.139, 195.359>
} else {
  if (t > 7772.91) {
    if (left[1] < 167.104) {
      y = 0.595663*pow(left[3],1)
          +0.584884*pow(right[2],1)
```

```
            -0.14832*pow(left[2],1)
            -0.646768*pow(right[1],1)
            +0.375642*pow(left[1],1)
            +0.477733*pow(right[0],1)
            -0.185438*pow(left[0],1)
            -6.5451;
        //(11317, 1434) <219.241, 215.915>
    } else {
        y = 0.616402*pow(left[3],1)
            +0.603696*pow(right[2],1)
            -0.18247*pow(left[2],1)
            -0.623408*pow(right[1],1)
            +0.39671*pow(left[1],1)
            +0.452291*pow(right[0],1)
            -0.199426*pow(left[0],1)
            -8.13588;
        //(1241, 133) <211.091, 152.021>
    }
} else {
    y = 0.716837*pow(left[3],1)
        +0.639319*pow(right[2],1)
        -0.394613*pow(left[2],1)
        -0.560577*pow(right[1],1)
        +0.459409*pow(left[1],1)
        +0.430266*pow(right[0],1)
        -0.211955*pow(left[0],1)
        -9.64711;
    //(6230, 779) <173.144, 164.914>
}
}
TMSE: 202.083
VMSE: 195.358
GMSE: 189.556
```

Essentially, the rules consist of linear blends of the left and right channels to obtain the predicted sample.

- The best model tree describing the **Time-series** dataset is as follows:

```
if (t2 < -0.0185124) {
```

```
if (t1 < 0.0263429) {
  if (t0 > 0.00343757) {
    y = -2.63082*pow(t0,1)
        +0.972257*pow(t1,1)*pow(t2,1)
        -0.0932611*pow(t2,1)
        +0.326075;
    //(101, 12) <0.123809, 0.0717015>
  } else {
    y = 1.00471*pow(t0,1)*pow(t1,1)*pow(t2,1)
        +0.669822;
    //(97, 8) <0.102582, 0.037943>
  }
 } else {
  y = -0.19592*pow(t1,1)
      +0.979469*pow(t2,1)
      +0.342466;
  //(193, 25) <0.0924138, 0.0926662>
 }
} else {
  y = 0.305796*pow(t1,1)
      -1.39869*pow(t2,2)
      +1.4918;
  //(409, 55) <0.0823577, 0.0943271>
}
TMSE: 0.0924692
VMSE: 0.086686
GMSE: 0.0857686
```

What is interesting to note, is that the model tree is almost identical to the generating function of section 4.4.1.

From the best solutions listed above, the maximum utilised polynomial order was 9 (for the house-16H dataset). This suggests that a maximum polynomial order of larger than 9 will result in no improvement in generalisation performance. For a large proportion of the above solutions, the utilised polynomial order was no greater than 3. This indicates that cubic surfaces sufficiently describe most databases, including time-series.

For some of the above results, outliers in the dataset were detected and isolated by rules. This indicates that there is still redundancy to be removed from the model tree solutions. Also, the removal of these outliers will improve the generalisation accuracy of the models.

These outliers should be removed by some heuristic, *e.g.* rules that cover a smaller number of patterns than some threshold should be removed and the patterns covered by the rules should be discarded from the training set. Thus, further improvements in accuracy and complexity are possible.

4.5 CONCLUSION

This chapter discussed a genetic program for the mining of continuous-valued classes (GPMCC). The performance of the GPMCC was evaluated against other algorithms such as Cubist and NeuroLinear for a wide variety of problems. Although the generalisation ability of the GPMCC method was slightly worse than the other methods, the complexity and number of generated rules were significantly smaller than that of other methods. The GPMCC method was also fairly robust, in that the parameter choices did not significantly effect any outcomes of the GPMCC method. The success of the GPMCC method can be attributed to the specialised mutation and crossovers, and can also be attributed to clustering. Another important aspect to the GPMCC method is the development of a fragment pool, which served as a belief space for the genetic program. The fragments of the fragment pool resulted in structurally optimal models for the terminal nodes of the GPMCC method. The fitness function was also crucial, because it penalised chromosomes with a high level of complexity.

Although the genetic program presented in this chapter seems to be fairly effective both in terms of rule accuracy and complexity, the algorithm was not particularly fast. The speed of the algorithm is seriously affected by the recursive procedures used to perform fitness evaluation, crossover and mutation on the chromosomes (model trees). This problem can be solved in two ways: implement a model tree as an array or change the model tree representation to a production system.

If the model trees of the genetic program are represented as an array, clever indexing of the array will negate the need for any recursive functions. However, the array would have to represent a full binary tree which could unnecessarily waste system memory if the model trees are sparse. If the model tree representation is changed to a production system, the mutation and crossover operators will have to be re-investigated.

Envisioned future developments to the GPMCC method include the revision of the attribute

tests. These attribute tests could be revised to implement non-linear separation boundaries between continuous classes. The GASOPE method could then be used to efficiently approximate these non-linear separation boundaries. The fragment pool discussed in this chapter could be used to implement a function set for the separation boundaries, in a manner similar to that of the terminal set.

The next chapter presents the conclusion to this thesis.

Chapter 5

CONCLUSION

This chapter briefly summarises the findings and contributions of this thesis, followed by a discussion of directions for future research

5.1 SUMMARY

Data mining is becoming increasingly important as a means of automating and enhancing traditional knowledge discovery processes. In order to completely satisfy the four objectives of data mining, *i.e.* accuracy, comprehensibility, crispness and novelty, new tools need to be developed. However, the development of new tools should also satisfy a number of qualitative requirements, *i.e.* scalability, efficiency, reliability *etc.*. The mining of continuous-valued classes provides its own unique challenges, evident in the relatively small number of data mining algorithms that exist for the mining of continuous classes.

This thesis developed a new genetic programming approach for the mining of continuous classes (GPMCC). Essentially, the GPMCC method evolved model trees in order to describe a data set, which was made up of patterns with continuous targets. The models for the model trees were obtained from a fast, efficient genetic algorithm that evolved structurally optimal polynomial expressions (GASOPE). Both the GASOPE and the GPMCC method utilised a fast, rough clustering algorithm in order to reduce the search space.

The GASOPE method evolves a population of individuals, which represent polynomial expressions. Specifically, the GASOPE method optimises the structure of the polynomial expressions, *i.e.* the GASOPE method determines the best term architecture, and the discrete

least squares method is employed to obtain the coefficients of the terms in an expression. A modified k-means clustering algorithm is utilised by the GASOPE method to continually draw stratified random samples of the training patterns during the learning process. This stratified random sampling strategy drastically reduces the computation time required by the method.

The GPMCC method evolves a population of individuals, which represent model trees. The models are obtained from a fragment pool, which implements a belief space of terminal symbols. The fragment pool implements mutation and crossover operators to adjust this belief space periodically. An iterative learning strategy is utilised by the GPMCC method in order to reduce the number of training patterns presented to the GPMCC learning process. In addition, the iterative learning strategy results in increased generalisation accuracy.

Experimentally, the GASOPE method was compared to a standard back-propagation neural network, which implemented gradient descent. The GASOPE method performed significantly better than the neural network both in terms of generalisation accuracy and training time. In fact, the training time of the GASOPE method was orders of magnitude faster than that of the neural network. The GASOPE method was also shown to produce the best approximating polynomial structure for a given data set.

The GPMCC method was compared to both NeuroLinear and Cubist. Although the GPMCC method was not significantly less accurate than both NeuroLinear and Cubist, the GPMCC method was shown to generate a substantially smaller number of rules. The rules generated by the GPMCC method were also less complex than those of the other methods. Overall, the GPMCC method proved to be a very capable tool for the mining of continuous-valued classes.

5.2 FUTURE RESEARCH

Throughout this thesis a number of new directions for future research presented themselves. These ideas are briefly summarised below:

Adaptive parameter settings for the GASOPE and GPMCC methods

Both the GASOPE and the GPMCC methods of chapters 3 and 4 introduce a number of new parameters, some of which were determined experimentally to be fairly robust. These parameters can be directly optimised using the evolutionary strategies approach of section 2.2.3. This

approach, however, may be a relatively time-consuming undertaking.

GASOPE representation adjustments

As was mentioned in section 3.5, polynomial structures are poor predictors of periodic data, particularly when applied to extrapolation tasks. This problem can be solved in one of two ways: Build an expression that utilises a periodic function such as cosine or sine, or use only linear predictors at the ends of the approximation interval.

Performance of the GPMCC method

The speed of the GPMCC method is seriously affected by the recursive procedures used to perform fitness evaluation, crossover and mutation on the chromosomes (model trees). This problem can be solved in two ways: implement a model tree as an array or change the model tree representation to a production system.

GPMCC attribute test revision

The attribute tests of the GPMCC method could be revised to implement non-linear separation boundaries between continuous classes. The GASOPE method could then be used to efficiently approximate these non-linear separation boundaries. The fragment pool discussed in section 4.3.3 could be used to implement a function set for the separation boundaries, in a manner similar to that of the terminal set.

Bibliography

[1] H.A. Abass, R.A. Saker, and C.S. Newton, editors. *Data Mining: A Heuristic Approach.* Idea Publishing Group, 2002.

[2] G. Adomavicius and A. Tuzhilin. Using data mining methods to build customer profiles. *Computer*, 34(2):74–82, 2001.

[3] E. Alba, J.F. Aldana, and J.M. Troya. Genetic algorithms as heuristics for optimizing ANN design. In R.F. Albrecht, C.R. Reeves, and N.C. Steele, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 683–690. Springer-Verlag, 1993.

[4] K. Alsabti, S. Ranka, and V. Singh. An efficient parallel algorithm for high dimensional similarity join. In *IPPS: 11th International Parallel Processing Symposium*. IEEE Computer Society Press, 1998.

[5] M.R. Anderberg. *Cluster Analysis for Applications.* Academic Press, New York, 1973.

[6] P.J. Angeline. Evolving predictors for chaotic time series. In S. Rogers, D. Fogel, J. Bezdek, and B. Bosacchi, editors, *Proceedings of SPIE (Volume 3390): Application and Science of Computational Intelligence*, pages 170–180, Bellingham, Washington, 1998.

[7] T. Bäck, D.B. Fogel, and T. Michalewicz, editors. *Evolutionary Computation 1: Advanced Algorithms and Operators.* IOP Press, 2000.

[8] T. Bäck, D.B. Fogel, and T. Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators.* IOP Press, 2000.

[9] G. Ball and D. Hall. A clustering technique for summarizing multivariate data. *Behavioral Science*, 12:153–155, 1967.

[10] E. Basson and A.P. Engelbrecht. Approximation of a function and its derivatives in feed-forward neural networks. In *IEEE International Joint Conference on Neural Networks.* paper 2152, Washington D.C., 1999.

[11] M.L. Beretta, D.G. Antoni, and A.G.B. Tettamanzi. Evolutionary synthesis of a fuzzy image compression algorithm. In *Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing*, volume 1, pages 466–470, Aachen, Germany, September 1996. Verlag Mainz, Aachen (Germany).

[12] J. Biles. Genjam: A genetic algorithm for generating jazz solos. In *The Computer Music Association, Proceedings of the International Computer Music Council*, 1994.

[13] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1992.

[14] C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, 1998. http://www.ics.uci.edu/~mlearnMLRepository.

[15] J. Bridle. Alphanets: A recurrent neural network architecture with a hidden markov model interpretation. *Speech Communication*, 9(1):83–92, 1990.

[16] R.L. Burden and J.D. Faires. *Numerical Analysis, 6th Edition*. Brooks/Cole Publishing Company, 1997.

[17] N. Cesa-Bianchi, P. Long, and M. Warmuth. Worst-case quadratic loss bounds for a generalization of the widrow-hoff rule. In *Proceedings of the 6th Annual Workshop on Computer Learning Theory*, pages 429–438, New York, 1993. ACM Press.

[18] Y. Chen, Z. Nakao, and X. Fang. A parallel genetic algorithm based on the island model for image restoration. In *Proceedings of the 1996 IEEE Signal Processing Society Workshop*, pages 109–118, Kyoto, September 1996. IEEE Press.

[19] T. Cibas, F. Fogelman Soulié, P. Gallinari, and S. Raudys. Variable selection with neural networks. *Neurocomputing*, 12:223–248, 1996.

[20] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *Proceedings of the Fifth European Working Session on Learning*, pages 151–163, Berlin, 1991. Springer.

[21] P. Clark and T. Niblitt. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[22] I. Cloete and J. Ludik. Increased complexity training. In *Proceedings of the International Workshop on Artificial Neural Networks*, pages 267–271, Berlin, 1993. Springer-Verlag.

[23] I. Cloete and J. Ludik. Delta training strategies. In *IEEE World Congress on Computational Intelligence, Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 295–298, Orlando, June 1994.

[24] I. Cloete and J. Ludik. Incremental training strategies. In *International Conference on Artificial Neural Networks*, volume 2, pages 743–746, Sorrento, Italy, May 1994.

[25] D.A. Cohn. Neural network exploration using optimal experiment design. In *AI Memo No 1491, Artificial Intelligence Laboratory, Massachusetts Institute of Technology*, 1994.

[26] W. Daelemans, P. Berck, and S. Gillis. Linguistics as data mining: Dutch diminutives. In T. Andernach, M. Moll, and A. Nijholt, editors, *CLIN V, Papers from the Fifth Computational Linguistics in the Netherlands Meeting*, pages 59–72, 1995.

[27] C.R. Darwin. *On the Origin of Species*. John Murray, London, 1859.

[28] G.J. Deboeck and T.K. Kohonen, editors. *Visual Explorations in Finance: With Self-Organizing Maps*. Springer Verlag, 1998.

[29] J.E. Dennis (Jr) and R.E. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice Halls, Englewood Cliffs, New Jersey, 1983.

[30] J. Eggermont, A.E. Eiben, and J.I. van Hemert. Adapting the fitness function in GP for data mining. In R. Poli, P. Nordin, W.B. Langdon, and T.C. Fogarty, editors, *Genetic*

Programming, Proceedings of EuroGP'99, volume 1598, pages 193–202, Goteborg, Sweden, 26-27 1999. Springer-Verlag.

[31] A.P. Engelbrecht. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Transactions on Neural Networks*, 12(6):1386–1399, 2001.

[32] A.P. Engelbrecht and R. Brits. Supervised training using an unsupervised approach to active learning. In *Neural Processing Letters*, pages 247–269. Kluwer Academic Publishers, Netherlands, 2002.

[33] A.P. Engelbrecht and I. Cloete. Feature extraction from feedforward neural networks using sensitivity analysis. In *International Conference on Systems, Signals, Control and Computers*, volume 2, pages 221–225, Durban, South Africa, 1998.

[34] A.P. Engelbrecht and I. Cloete. Selective learning using sensitivity analysis. In *IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks*, pages 1150–1155, Anchorage, Alaska, 1998.

[35] A.P. Engelbrecht and I. Cloete. Incremental learning using sensitivity analysis. In *IEEE International Joint Conference on Neural Networks*, Washington D.C., 1999. paper 380.

[36] S.S. Epp. *Discrete Mathematics with Applications, Second Edition.* Brooks / Cole Publishing Company, 1995.

[37] K. Fanghanel, R. Hein, K. Köllmann, and H.C. Zeidler. Optimizing wavelet transform coding using a neural network. In *Proceedings of the IEEE International Conference on Information, Communications and Signal Processing*, volume 3, pages 1341–1343, Singapore, September 1997.

[38] L. Fletcher, V. Katkovnik, F.E. Steffens, and A.P. Engelbrecht. Optimizing the number of hidden nodes of a feedforward artificial neural network. In *IEEE World Congress on Computational Intelligence, In proceedings of the International Joint Conference on Neural Networks*, pages 1608–1612, Anchorage, Alaska, 1998.

[39] D. B. Fogel. Evolutionary programming: an introduction and some current directions. *Statistics and Computing 4*, pages 113–129, 1994.

[40] J.B. Fraleigh and R.A. Beauregard. *Linear Algebra, 3rd Edition*. Addison-Wesley Publishing Company, 1995.

[41] A.A. Freitas. A genetic programming framework for two data mining tasks: Classification and generalized rule induction. In J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, and R.L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 96–101, Stanford University, 13-16 1997. Morgan Kaufmann.

[42] B. Fritzke. Incremental learning of local linear mappings. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 217–222, Paris, October 1995.

[43] K. Fukumizu. Active learning in multilayer perceptrons. In M.C. Mozer and M.E. Hasselmo, editors, *Advances in Neural Information Processing*, volume 8, pages 295–301, 1996.

[44] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[45] Y.L. Geom. Genetic recursive regression for modelling and forecasting real-world chaotic time series. *Advances in Genetic Programming*, 3:401–423, 1999.

[46] F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural network architectures. *Neural Computation*, 7:219–269, 1995.

[47] N. Göckel, G. Pudelko, R. Drechsler, and B. Becker. A hybrid genetic algorithm for the channel routing problem. In *Proceedings of the International Symposium on Circuits and Systems*, volume 2, pages 675–678, 1996.

[48] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, 1989.

[49] G.E. Goldberg and K. Deb. A comparitive analysis of selection schemes used in genetic algorithms. In G.J.E Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan-Kaufman, 1991.

[50] R. Haggarty. *Fundamentals of Mathematical Analysis, 2nd Edition*. Addison-Wesley Publishing Company, 1993.

[51] J. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.

[52] B. Hassibi and D.G. Stork. Second order derivatives for network pruning. In C. Lee Giles, S.J. Hanson, and J.D. Cowan, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 164–171, 1993.

[53] R. Hinterding, H. Gielewski, and T.C. Peachey. The nature of mutation in genetic algorithms. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 65–72, San Francisco, 1995. Morgan Kaufmann.

[54] Y. Hirose, K. Yamashita, and S. Hijiya. Back-propagation algorithm which varies the number of hidden units. *Neural Networks*, 8:1277–1299, 1996.

[55] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[56] L. Holmström and P. Koistinen. Using additive noise in back-propagation training. *IEEE Transactions on Neural Networks*, 3(1):24–38, 1992.

[57] K. Hornik. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[58] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3:551–560, 1990.

[59] S. Horowitz and T. Pavlidis. Picture segmentation by a tree traversal algorithm. *Journal of the Association for Computing Machinery*, 23(2):368–388, April 1976.

[60] A. Iserles. Generalized leapfrog methods. *IMA Journal of Numerical Analysis*, 6:381–392, 1986.

[61] A. Ismail and A.P. Engelbrecht. Training product units in feedforward neural networks using particle swarm optimization. In V.B. Bajic and D. Sha, editors, *Development and*

Practice of Articial Intelligence Techniques, Proceedings of the International Conference on Articial Intelligence, pages 36–40, Durban, South Africa, 1999.

[62] M.A. Kaboudan. Genetic evolution of regression models for business and economic forecasting. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1260–1268. IEEE Press, 1999.

[63] R. Kamimura and S. Nakanishi. Weight decay as a process of redundancy reduction. *World Congress on Neural Networks*, 3:486–489, 1994.

[64] S. Kaski. *Data Exploration using Self-Organizing Maps.* PhD thesis, Laboratory of Computer and Information Science, Department of Computer Science and Engineering, Helsinki University of Technology, 1997.

[65] T. Kohonen. *Self-Organization and Associative Memory, 3rd Edition.* Springer-Verlag, Berlin, Germany, 1989.

[66] T. Kohonen. *Self-Organizing Maps.* Springer-Verlag, Berlin, 1995.

[67] J.R. Koza. Genetic evolution and co-evolution of game strategies. In *The International Conference on Game Theory and Its Applications*, Stony Brook, New York, July 1992.

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

[69] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Howard, and L.D. Jackel. Handwritten digit recognition with a back-propagation network. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems, volume 2*, pages 396–404. Morgan Kaufmann, San Mateo, 1990.

[70] Y. Le Cun, J. Denker, S. Solla, R.E. Howard, and L.D. Jackel. Optimal brain damage. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605, San Mateo, 1990. Morgan Kauffman.

[71] D.J.C. MacKay. *Bayesian Methods for Adaptive Models.* PhD thesis, California Institute of Technology, 1992.

[72] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L.M. Le Cun and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1: Statistics, pages 281–297, Berkeley and Los Angeles, 1967. University of California Press.

[73] R.E. Marmelstein and G. Lamont. Pattern classification using a hybrid genetic program-decision tree approach. In J.R. Koza, editor, *Genetic Programming 98: Proceedings of the Third International Conference*, pages 223–231. Morgan Kaufmann, 1998.

[74] J.L. McClelland and D.E. Rumelhart. Training hidden units: The generalized delta rule. In *Explorations in Parallel Distributed Processing*, volume 3, chapter 5, pages 121–159. MIT Press, 1988.

[75] M.F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.

[76] N. Nikolaev and H. Iba. Genetic programming using chebishev polynomials. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 89–96, San Francisco, 2001. Morgan Kaufmann Publishers.

[77] N.J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, Inc., San Fransisco, 1998.

[78] N. Ohnishi, A. Okamoto, and N. Sugie. Selective presentation of learning samples for efficient learning in multi-layer perceptron. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 1, pages 688–691, Washington D.C., January 1990.

[79] Y.H. Pao and Y. Takefuji. Functional-link net computing: Theory, system architecture, and functionalities. *Computer*, 25(5):76–79, May 1992.

[80] W.M. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical recipes: The art of scientific computing*. Cambridge University Press, 1986.

[81] J.R. Quinlan. Learning efficient classification procedures and their application to chess endgames. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine*

Learning: An Artificial Intelligence Approach, volume 1, pages 463–482. Tioga Press, Palo Alto, 1983.

[82] J.R. Quinlan. Learning with continuous classes. In Adams and Sterling, editors, *Proceedings of Artificial Intelligence '92*, pages 343–348, Singapore, 1992. World Scientific.

[83] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, 1993.

[84] J.R. Quinlan. Rulequest research data mining tools. RuleQuest Research Pty Ltd, Australia, 2002. http://www.rulequest.com/see5-info.html.

[85] I. Rechenberg. Evolution strategy. In J.M Zurada, R. Marks II, and C. Robinson, editors, *Computational Intelligence - Imitating Life*, pages 147–159. IEEE Press, 1994.

[86] R.G. Reynolds. An introduction to cultural algorithms. In *Proceedings of the Third Annual Conference on Evolutionary Computing*, pages 131–139, 1994.

[87] C. Sage. An overview of radiofrequency/microwave radiation studies relevent to wireless communications and data. In *Procceedings of the International Conference on Cell Tower Siting*, pages 90–105, Salzburg, Austria, June 2000.

[88] D. Salomon. *Data Compression*. Springer, 2nd edition, 2000.

[89] F. Schweitzer, H. Rose, W. Ebeling, and O. Weiss. Optimization of road networks using evolutionary strategies. *Evolutionary Computation*, 5(4):419–438, 1997.

[90] R.S. Sellar, M.A. Stelmack, S.M. Batill, and J.E. Renaud. Response surface approximations for discipline coordination in multidisciplinary design optimization. In *American Institute of Aeronautics and Astronautics*, number 96 in 1383, 1996.

[91] R. Setiono. Generating linear regression rules from neural networks using local least squares approximation. In J. Mira and A. Prieto, editors, *Connectionist Model of Neurons, Learning Processes, and Artificial Intelligence, Proceedings of the* 6^{th} *International Work-Conference on Artificial and Natural Neural Networks*, volume 1, pages 277–284, Granada, Spain, June 2001. Springer.

[92] R. Setiono and L.C.K. Hui. Use of quasi-newton method in a feedforward neural network construction algorithm. *IEEE Transactions on Neural Networks*, 6(1):273–277, 1995.

[93] R. Setiono and W.K. Leow. Pruned neural networks for regression. In R. Mizoguchi and J. Staney, editors, *Proceedings of the 6th Pacific Rim Conference on Artificial Intelligence, PRICAI 2000, Lecture Notes in AI 1886*, pages 500–509, Melbourne, Australia, 2000. Springer.

[94] R. Setiono, W.K. Leow, and J.M. Zurada. Extraction of rules from artificial neural networks for nonlinear regression. *IEEE Transactions on Neural Networks*, 13(3):564–577, 2002.

[95] E. Spertus. Smokey: Automatic recognition of hostile messages. In *Proceedings of Innovative Applications of Artificial Intelligence (IAAI)*, pages 1058–1065, 1997.

[96] A.G.W. Steyn, C.F. Smit, S.H.V. du Toit, and C. Strasheim. *Modern Statistics in Practice*. J. L. van Schaik, 1996.

[97] C.M. Vest. Biotech research presents a major opportunity for massachusetts economy. Boston Sunday Globe, 11 August 2002.

[98] A.S. Weigend, D.E. Rumelhart, and B.A. Huberman. Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing Systems*, volume 3, pages 875–882, 1991.

[99] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[100] L.F.A. Wessels and E. Barnard. Avoiding false local minima by proper initialization of connections. *IEEE Transactions on Neural Networks*, 3(6):899–905, 1992.

[101] P.M. Williams. Bayesian regularization and pruning using a laplace prior. *Neural Computation*, 7:117–143, 1995.

[102] S.W. Wilson. Function approximation with a classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 974–981, San Francisco, 2001. Morgan Kaufmann Publishers.

[103] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[104] J. Yao, Y. Li, and C. Tan. Forecasting the exchange rates of CHF vs USD using neural networks. In *Journal of Computational Intelligence in Finance*, volume 5, pages 7–13, 1997.

[105] Z. Zheng. *Constructing New Attributes for Decision Tree Learning*. PhD thesis, Basser Department of Computer Science, The University of Sydney, Australia, 1996.

[106] J. M. Zurada. *Introduction to Artificial Neural Systems*. PWS Publishing Company, 1992.

Appendix A

SYMBOLS

Tables A.1 and A.2 list and define the symbols used throughout this thesis.

Table A.1: Table of symbols

Symbol	Meaning
a	Input attribute
b	Target output
$\mathbf{b}$	Vector of target outputs
$b^{\bullet}$	Predicted output
c	Number of classes
d	Model complexity
$connections$	Artificial neuron inputs
f_{AN}	Artificial neuron activation function
f'_{AN}	Artificial neuron activation function derivative
g	Evolutionary computing generation counter
h	Artificial neural network hidden unit iterator
i	Artificial neural network input unit iterator or pattern iterator
j	Term iterator
k	Number of clusters, classes, coefficients or complexity (clear from context)
l	Pattern iterator
m	Total number of input attributes
n	Generic maximum, mostly used to mean the maximum order of a polynomial
net	Artificial neuron weighted sum variable
o	Test outcomes
p	Maximum number of terms
r	Real-valued coefficient
$\mathbf{r}$	Vector of real-valued coefficients
s	Sample size
t	Artificial neural network output unit iterator
v	Possible attribute value
$\mathbf{v}$	Artificial neural network output weight vector
w	Artificial neural network weight
$\mathbf{w}$	Cluster centroid vector or artificial neural network hidden weight vector
x	Input value
y	Target output
$y^{\bullet}$	Predicted output
A	Attribute or Attribute matrix (clear from context)
C_{δ}	Cluster
C_{ψ}	Class
$C_{subscript}$	Antecedent term
E	Error term
E_{MA}	Mean absolute error
E_{MS}	Mean squared error
E_{Q}	Quantisation error
E_{SS}	Sum squared error
F_{ω}	Fragment in a fragment pool
G	Evolutionary computing population
G_{GA}	Genetic algorithm population
G_{FP}	Fragment pool
G_{GP}	Genetic program population

Table A.2: Table of symbols (cont.)

Symbol	Meaning
H	Total number of hidden units in an artificial neural network
I	Total number of input units in an artificial neural network
I_α	Parent individual to be involved in crossover
I_β	Parent individual to be involved in crossover
I_γ	Child individual resulting from crossover
I_ω	Individual in a GASOPE population or fragment pool
I_χ	Individual in a GPMCC population
M	Maximum number of generations
L	3-Piece piecewise linear approximation
N	Maximum number of individuals
N_χ	An arbitrary node in I_χ
O	Evolutionary computing temporary population
P	Training set or penalty term (clear from context)
Q	A set of patterns belonging to some case
R	A set of patterns covered by a rule
S	Sample set
T	Total number of output units in an artificial neural network
T_ξ	One of the sets of term-coefficient mappings in and individual I_ω
$U_{subscript}$	User defined parameter
X	Test
α	Artificial neural network momentum term
$\alpha_{subscript}$	Intercept
$\beta_{subscript}$	Gradient
δ	Cluster iterator
ε	Artificial neural network epochs iterator
$\varepsilon_{subscript}$	Penalty terms
η	Artificial neural network learning rate
θ	Artificial neuron threshold
λ	Control parameter for the sigmoid function
$\lambda_{subscript}$	Natural valued order
ξ	Term-coefficient iterator
σ	Artificial neural network error signal
τ	Input attribute iterator or bias (clear from context)
ρ_l	Pattern
ϕ	Outcome iterator
χ	Population iterator for a genetic program
ψ	Class iterator
ω	Individual iterator

Index