Automatic Feature Extraction for Pattern Recognition

by

Jamie Sherrah

Thesis submitted for the degree of

Doctor of Philosophy

SUB CRUCE LUMEN

Department of Electrical and Electronic Engineering
Faculty of Engineering
The University of Adelaide
South Australia

10 July 1998

Contents

Corrigenda

Jamie Sherrah

November 18, 1998

Position	Correction
page 131, paragraph 2	"can set" becomes "can be set"
page 177, paragraph 6	"examination **diabetes**" becomes "examination of **diabetes**"
page 187, paragraph 4	"was when" becomes "when"

List of Figures

List of Tables

Abstract

A typical pattern recognition system consists of two stages: the pre-processing stage to extract features from the data, and the classification stage to assign the feature vector to one of several classes. While many general classifiers exist and are well-understood, the pre-processing stage is usually ad-hoc and designed by hand. Although the accuracy of the classifier is heavily dependent on the choice of features, there is little more guidance in the process of manual feature extraction than intuition, experience and trial-and-error.

To achieve automatic and near-optimal pre-processor design, a framework is required for the problem-independent extraction of features. Within such a framework, the concept of an optimal pre-processor can be formulated. The framework must allow pre-processors which are universally applicable and realisable using finite resources. Those frameworks already in existence, such as principal-component analysis and multi-layer perceptrons, are either unable to cope with arbitrary non-linearity or unable to be implemented using finite resources because they employ one type of constituent function and have a fixed structure.

In this thesis, a framework for automatic feature extraction is proposed, called the *generalised pre-processor*. This is an arbitrarily-interconnected feed-forward network with arbitrary non-linear functions at the nodes. The use of different constituent functions and irregular inter-connection strategies allows for the economic realisation of a pre-processor in more situations than the more uniform universal approximators, such as the multi-layer perceptron. A software system called the *Evolutionary Pre-Processor* is presented which performs a search over the space of generalised pre-processors. The system is used for supervised classification, and must be provided with a data set of measurement vectors and associated class labels. Based on genetic programming, the evolutionary pre-processor begins with a population of randomly-generated pre-processors. The fitness of each pre-processor is based on the estimated misclassification cost of a classifier trained on the pre-processed data. Through fitness-proportionate reproduction and recombination, the ability of the pre-processors to separate the data increases with generations.

The evolutionary pre-processor has been tested on 15 real and synthetic public-domain data sets. Neural networks, decision trees and five simple statistical classification techniques were applied to the same problems, and the results compared. The results show that the evolutionary pre-processor maintains good classification and generalisation performance, and is more accurate on average than the decision tree method. The neural network achieved the lowest classification errors on average, but was surpassed by the evolutionary pre-processor on some synthetic problems. Both the evolutionary pre-processor and the decision tree produce solutions which can be understood and interpreted by the user. These results must be considered with care, however, as they fluctuate with different random seeds and partitioning of the data.

The investigations of this thesis have revealed that a search over pre-processors is feasible. The synthesis of pre-processors from a variety of non-linear, and even discontinuous functions occasionally provides better discrimination than existing methods of classification, but for most problems gradient-descent methods are adequate. The evolutionary pre-processor has advantages for knowledge discovery due to the versatility with which appropriate functions can be combined, but is limited due to the high variability in results. It should be used in conjunction with other methods of knowledge discovery for reliable results. The evolved pre-processors and simple classifiers used by EPrep result in relatively accurate classification systems that can be implemented more economically than other methods.

Statement of Originality

I hereby declare that this work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and that, to the best of my knowledge and belief, it contains no material previously published or written by another person, except where due reference has been made in the text. I also give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying.

Jamie Sherrah

Signed:

Dated: 8th July 1998

Acknowledgments

My heart-felt gratitude goes to my two supervisors and friends, Robert Bogner and Salim Bouzerdoum. Their guidance, technical assistance, encouragement and availability has made this thesis possible. Thanks go to my colleagues Carmine Pontecorvo, Steven Wawryk and Ben Raymond, who have been daily participants in my work, albeit indirectly. Acknowledgment and thanks must go to the behind-the-scenes workers: my wife Kate, and my parents. Most of all, I want to thank my Lord Jesus Christ, without whom I can do nothing.

My thanks go to Daniel McMichael for his advice and rigour. The CSSIP Melbourne group provided useful feed-back on the work. Thanks to Wei-Yin Loh and Yu-Shan Shih for their prompt assistance with the QUEST software. Thanks also to the anonymous reviewers from the GP'97 conference.

This thesis was prepared using LaTeX 2$_{\varepsilon}$. The TreeTeX package (Brüggemann-Klein and Wood, 1989) was used to produce the QUEST trees. Another package named treetex (Blösch, 1993) was used to lay out the EPrep features.

Related Publications

Jamie R. Sherrah and Ravi Jain. "Classification of Heart Disease Data using the Evolutionary Pre-Processor". To appear in *Proceedings of the Engineering Mathematics and Applications Conference*, Adelaide University, July 1998.

Jamie R. Sherrah, Robert E. Bogner, and Abdesselam Bouzerdoum. "The Evolutionary Pre-Processor: Automatic Feature Extraction for Supervised Classification using Genetic Programming". In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 304-312, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

Jamie R. Sherrah "Automatic Feature Extraction using Genetic Programming" In John R. Koza, editor, *Late Breaking Papers at the 1997 Genetic Programming Conference*, page 298, Stanford University, CA, USA, 13-16 July 1997. Stanford Bookstore.

Jamie R. Sherrah, Robert E. Bogner, and Abdesselam Bouzerdoum. "Automatic Selection of Features for Classification using Genetic Programming". In V. L. Narasimhan and L. C. Jain, editors, *Proceedings of the Australian New Zealand Conference on Intelligent Information Systems*, pages 284-287. IEEE Press, November 1996.

Glossary

Abbreviations with page of first appearance, listed in alphabetical order.

ANN	22	artificial neural network
BOG	104	best-of-generation
BOR	104	best-of-run
BRACE	108	blocking RACE
CV	14	cross-validation
DSS	116	dynamic sub-set selection
EA	122	evolutionary algorithm
EDI	125	explicitly defined intron
EP	55	evolutionary programming
EPrep	97	evolutionary pre-processor
ES	54	evolution strategies
FSM	55	finite state machine
GA	53	genetic algorithm
GL	128	generalisation loss stopping criterion
GLIM	23	generalised linear machine
GP	54	genetic program
GPP	80	generalised pre-processor
HC	118	hill-climbing
HTML	138	hyper-text mark-up language
kNN	21	*k*-nearest neighbours
LOC	191	lines of code
LOOCV	108	leave-one-out cross-validation
MDL	42	minimum description length
MDTM	140	minimum-distance-to-means
ML	140	maximum likelihood
MLP	25	multi-layer perceptron
MML	42	minimum message length
PCA	79	principal component analysis
pdf	17	probability density function
PPD	140	parallelepiped
QUEST	39	quick, unbiased, efficient statistical trees
RACE	108	statistical algorithm for model selection
RAT	104	rational allocation of trials
SA	118	simulated annealing
STGP	70	strongly-typed genetic program
TP	127	training progress stopping criterion

Frequently-used mathematical symbols with page of first appearance.

Δ	112	number of samples to examine on each iteration of the RAT algorithm
$\delta(H)$	62	defining length of schema H
γ	108	similarity tolerance for RAT algorithm
$\phi(.)$	22	neuron activation function
τ^*	60	take-over time
A	52	population of individuals
$B(n,p)$	14	binomial distribution with probability of success p and number of trials n
C	7	number of classes
$\mathcal{D}$	8	training set
D_c	68	maximum allowable depth of trees resulting from genetic operators
D_i	68	maximum depth of initial random trees
$E[.]$	11	expected value
Fi	100	the ith feature of an individual generated by EPrep
G	52	maximum number of generations
H	62	schema
$I(.)$	13	logic function, returns 1 if argument is true, false otherwise
L	60	length of chromosome
M	52	population size
M_{mp}	56	mating pool size
N	7	number of data samples
$N(\mu,\sigma)$	121	normal distribution with mean μ and standard deviation σ
$N(x,T)$	49	neighbourhood function of point x
N_o	103	number of samples used by EPrep for local optimisation and RAT training
N_{ops}	107	number of genetic operators used by EPrep
N_{rep}	102	number of individuals for reproduction
$\mathcal{O}(.)$	48	order notation
$P(.)$	10	a probability value
R	53	number of runs
S	59	tournament size
$\mathcal{T}$	13	hold-out test set
$U(a,b)$	107	uniform distribution on the range $[a,b]$
$\mathcal{V}$	14	validation set
$\mathcal{X}$	10	universe of all measurement vectors
Xi	100	the ith input variable used with EPrep
a	52	individual in a population
c	7	actual class label of data sample
c'	8	predicted class label of data sample
c^*	10	class label resulting from Bayes decision
d	7	dimensionality of input data
$\bar{f}(g)$	62	average fitness of population at generation g
$f(.)$	52	fitness function
g	62	generation
g_{sort}	118	turn-over time of training samples in EPrep
$h(\mathbf{x})$	8	classification rule to classify measurement vector $\mathbf{x}$
$m(H,g)$	62	number of representatives of schema H at generation g
n_{sort}	117	number of samples per class to periodically sort in EPrep
n_{tr}	17	number of training samples
n_{tst}	17	number of test samples
n_{val}	17	number of validation samples
$o(H)$	62	order of schema H

$p(.)$	10	a probability density function
p_c	56	probability of crossover
p_d	63	probability of destruction of a schema
p_m	56	probability of mutation
p_r	56	probability of reproduction
p_s	63	probability of survival of a schema
p_{inv}	127	probability of inversion
p_{ip}	69	probability of selecting an internal node for crossover
pmc	11	probability of misclassification
$\mathbf{x}$	7	input measurement vector
$\mathbf{y}$	8	feature vector

Chapter 1

Introduction and Overview

This is a thesis about pattern recognition, and in particular about the use of genetic programming to automatically extract features from data for supervised classification. This introductory chapter serves as a guide for the whole thesis, giving the reader a flavour of the background and motivation for the work, as well as a brief description of the experiments carried out. For the partially-interested reader, Section 1.5 contains an overview of the thesis so that pertinent chapters can be quickly identified.

1.1 Background to the Research

Pattern recognition refers to the use of computers to recognise and classify objects based on measurements of those objects. The measurements may be images, sound waveforms, demographics, etc. Within the topic of pattern recognition, supervised classification is concerned with the task of learning to classify new objects based on knowledge gained from observations of previously-encountered objects.

A system designed to perform supervised classification usually consists of two sequential stages: a feature extraction stage, and a classification stage. The feature extractor, or pre-processor, transforms the data to allow more successful classification by the second stage, the classifier. There are many different types of general-purpose classifiers that have been designed for use over a broad range of problems. In comparison, the pre-processing stage is generally problem-dependent, and is usually constructed by the designer using experience, intuition and trial-and-error. There is no existing methodology to guide the designer, so the concept of optimal feature extraction only exists within the context of the method used.

Automatic feature extraction methods seek to extract salient features from data independent of the problem domain. Examples of existing methods are principal component analysis and artificial neural networks. Although these existing methods provide a framework for automatic feature extraction, they are limited in their scope of application by the availability of a very small number of functions. For instance, the multi-layer perceptron uses only sigmoidal activation functions to pre-process the data. While theorems exist stating the sufficiency of these functions to approximate all continuous transformations, only existence of such a transformation is guaranteed, and it may not be realisable using finite resources.

1.2 The Thrust of this Thesis

A new framework for automatic feature extraction is proposed here: a *generalised pre-processor* (GPP), which is an arbitrary composition of different functions, which can be non-linear and discontinuous. The main question investigated in this work is:

How effective is a generalised pre-processor at extracting features from real data when compared with existing automatic feature extraction methods?

The primary route of investigation was to compare the classification performance of the generalised pre-processor method with that of other known classification and automatic feature-extraction methods. The comparison took place over a test-bed of 15 public-domain data sets, consisting of synthetic and real-world data.

We argue that the research topic and investigations are important for the field of pattern recognition. The generalised pre-processor represents a fully-general framework for automatic feature extraction, which is essential to:

1. avoid repeated manual feature extraction on new problems,
2. allow the concept of a globally-optimal realisable pre-processor,
3. identify common features of different problems,
4. discover previously-unknown structure in data, and
5. allow the extraction of features from data for which no *a priori* knowledge is available.

The investigations conducted are important for two reasons:

Feasibility The existence of an optimal realisable pre-processor does not promise the feasibility of its synthesis. Genetic programming was used to investigate the feasibility of a search for an optimal generalised pre-processor.

Practical Utility Synthetic examples of the usefulness of a generalised pre-processing stage are demonstrated in Chapter 4. Such examples can always be contrived, but the important issue is whether the generalised pre-processor is more useful than existing methods for real problems.

1.3 Contributions of this Thesis

This thesis contributes to the fields of pattern recognition and genetic programming. The following original contributions to the body of knowledge are made in this thesis:

1. A new framework for automatic feature extraction is proposed (Section 4.8).
2. Genetic programming is applied to general supervised classification of real-world data sets (Chapter 6).
3. Analysis showing that population-based search methods converge on solutions of the appropriate size faster than single-point search methods (Section 4.9).
4. A comparison of genetic programming with well-known classification techniques on 15 data sets (Chapter 6).
5. A concise overview of pattern recognition (Chapter 2).
6. A concise overview of evolutionary computation (Chapter 3).
7. The demonstration of the inadequacy of the multi-layer perceptron for classification of a synthetic data set (Section 4.6).
8. The application of the rational-allocation-of-trials algorithm to classification problems, with the necessary modifications (Section 5.5.1).
9. The use of the simplex algorithm for local optimisation in genetic programming (Section 5.7).

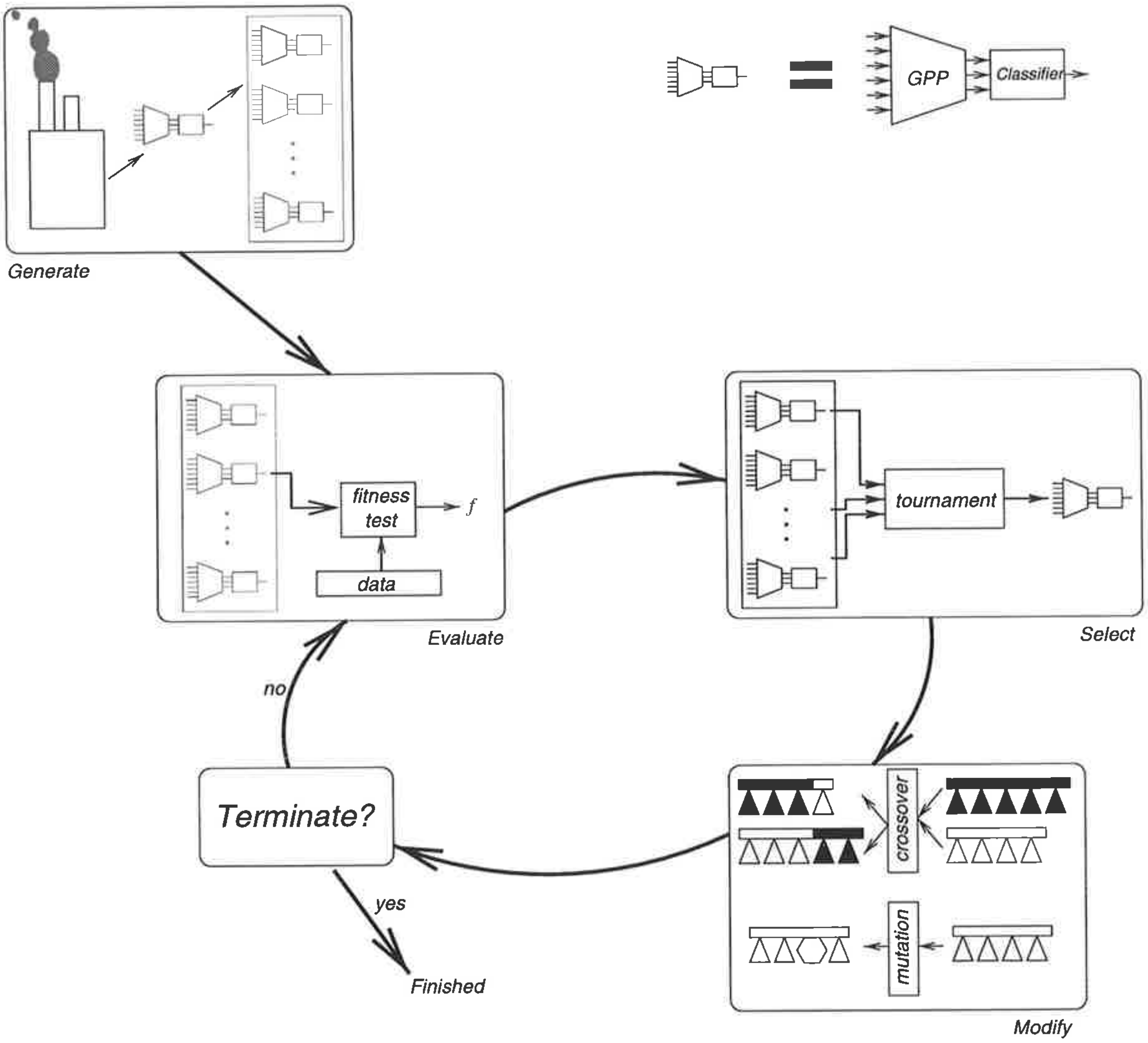


Figure 1.1: A high-level illustration of the evolutionary pre-processor algorithm.

1.4 Methodology

The method used to investigate the issues addressed in this thesis was empirical evaluation and comparison. The particular line of reasoning was to devise a method to perform a search over generalised pre-processors so that an optimal pre-processor could be sought. The result was the evolutionary pre-processor. Experiments were performed to evaluate the algorithm and compare it with existing methods for classification and automatic feature extraction.

The method used to search for an optimal generalised pre-processor needed to be able to arbitrarily combine different non-linear or discontinuous functions to form variable-length solutions. Genetic programming, an optimisation method based on the principles of biological evolution, was chosen as the basis for the evolutionary pre-processor algorithm. Figure 1.1 shows a high-level diagram of the execution of the evolutionary pre-processor. The algorithm performs a search by manipulating a population of individuals. Each individual is a solution to the problem, and consists of a generalised pre-processor and a classifier. The evolutionary pre-processor operates by iteratively modifying and re-combining the pre-processors in search of feature extractors that effectively transform the data for the classifier used.

The first stage in the algorithm randomly generates the initial population of solutions.

Each pre-processor is represented as a set of trees, each of which is a single feature. The classifier in each individual is one of a fixed set of simple well-known classification algorithms. The next stage evaluates the fitness of each individual, which quantifies the ability of the individual to classify the pre-processed training data. The fitness is an empirical estimate of classification error. Based upon these fitness values, the selection stage chooses individuals from the population to form the mating pool. An individual is selected by randomly choosing a set of individuals from the population to participate in a tournament. The individual in the tournament with the best fitness value is the winner of the tournament, and proceeds to the mating pool. Individuals from the mating pool are then stochastically modified by genetic operators to form new individuals. The modification may involve the combination of two individuals from the mating pool via the crossover operator, or the perturbation of a single feature from an individual by a mutation operator. The new individuals are placed into the new population, which is used for the next iteration of the algorithm. After the modification stage, the termination criteria are tested. The algorithm terminates if progress has stagnated, the best individuals are losing their generalisation capabilities, or a prescribed maximum number of iterations has been performed. Otherwise the process is repeated using the new population of solutions. Through natural selection, the population is iteratively transformed from a random mess to a set of individuals whose estimated classification error is lower than the the classifier used on the original data.

The evolutionary pre-processor was evaluated by running it on 15 public-domain data sets, most of which came from real-world problems. Having established that the search was able to find a pre-processor able to improve upon the performance of a classifier, the evolutionary pre-processor was then compared on the same data sets with several well-known classification methods: neural networks, decision trees, and five simple statistical methods.

1.5 Outline of this Thesis

This section contains a brief outline of the contents of the thesis. The work begins in Chapter 2 with an introduction to pattern recognition. The key topic for this thesis is supervised classification, which is the focus of Chapter 2. The classification methods used for the experimental work are described, and methods for measuring the performance of a classifier are explained. Methods for comparing classifiers are also presented.

Chapter 3 is an introduction to evolutionary computation. First, the topics of optimisation and heuristic search are introduced. Next, each of the four paradigms of evolutionary computation are briefly introduced, and their differences discussed. Then genetic algorithms and genetic programming are examined in greater detail, because these two approaches form the basis for the evolutionary pre-processor algorithm.

The main arguments of the thesis begin in Chapter 4. Existing methods of feature extraction are discussed, and their short-comings are pointed out. To address these short-comings, the generalised pre-processor is proposed as a framework for automatic feature extraction. A series of hypotheses are presented and argued for to establish the position taken in this thesis. To be specific, the hypotheses propose that the GPP method can be more appropriate than existing methods for feature extraction. It is then conjectured that population-based heuristic search techniques are more appropriate to search for an optimal GPP than search methods based on a single point. The two questions are then raised: is a search for an optimal GPP feasible for real problems, and is this approach useful for knowledge discovery? It is the task of the next two chapters to address these questions. The chapter ends with a review of previous work in the field.

The evolutionary pre-processor algorithm used to empirically investigate the research questions is fully described in Chapter 5. First, the choices of solution representation, population model and objective function are described and justified. Then an overview of the whole algorithm is presented. The rest of the chapter describes each portion of the algorithm in detail, including relevant background literature. The last section in the chapter presents some heuristics for selecting the algorithm parameters.

The experiments performed are described in Chapter 6, and the results are presented. The chapter begins with a description of the data sets used to test the algorithm. The two main experiments were the use of EPrep to classify the test data sets, and a comparison of EPrep with existing methods for classification and automatic feature extraction. The chapter next presents a description of the experimental configuration. The results of the EPrep algorithm on the test problems follow, and the behaviour of the algorithm is examined in detail. Next, the results of the existing algorithms are presented. Then a comparison is made between the algorithms and EPrep using statistical tests. The comparison is based not only on estimated misclassification rate, but also on interpretability of results and computational complexity.

The conclusion to the work is written in Chapter 7. The experimental results are linked back to the hypotheses and research questions presented in Chapter 4. Then conclusions are presented about the overall research topic, followed by implications for the larger field of research. The chapter ends with suggestions for future research.

There are five appendices. Appendix A contains instructions for the use of the CD-rom found at the back of this thesis, which contains the detailed reports generated by EPrep. Appendix B contains some details of the evolutionary pre-processor software, and a discussion of some of the major design issues of the implementation. Appendix C describes each of the experimental data sets in detail. Appendix D contains the parameter files used for the experiments of EPrep. Appendix E shows plots of the summarised results obtained by EPrep on the test problems.

1.6 Conclusion

This chapter has presented an overview of this thesis, on the basis of which the reader can proceed with confidence, or skip to particularly relevant sections. The background of the research problem was introduced, and the main question addressed by the work was presented. The basic methodology used for the research was described, and an outline of the remainder of the thesis was presented.

Chapter 2

Pattern Recognition

2.1 Introduction

This thesis presents novel work in pattern recognition with the use of genetic programming as a tool. To grasp the main concepts beginning in Chapter 4, a knowledge of pattern recognition and genetic programming is required. In this chapter, the field of pattern recognition is introduced and reviewed. Since pattern recognition is such a voluminous subject, attention has been restricted to those topics that are immediately relevant to the thesis. The reader familiar with pattern recognition and supervised classification may like to skip this chapter, referring back to it if needs be.

2.2 What is Pattern Recognition?

We humans are constantly engaged in pattern recognition. The predominant form of pattern recognition we engage ourselves in is through our visual senses; correspondingly about 60% of the brain's neurons are attributable to our visual system. As we look around us, we recognise scores of objects without an ounce of effort. Whether they occlude one another, appear at different angles of rotation or in different lighting conditions, our visual system enables us to perform this general and complex task subconsciously. Although we exist in a three-dimensional world and have two eyes to exploit this fact, we humans can still recognise objects in two-dimensional images. Consider the picture in Figure 2.1. We clearly recognise the candlestick, toothbrush, glass and other objects in the scene, regardless of occlusions or their pose. We also deduce that the toothbrush and toothpaste are *in* the glass, which is *on* the dresser.

Pattern recognition is not constrained to the visual senses alone. We can recognise a face, the voice of an acquaintance, detect facial similarities in two blood-related people, and identify a musician from their style. Specialists from many fields, such as mining and medicine, apply their experience and knowledge to detect and diagnose certain scenarios. Amazingly, all this is done automatically by our bodies without our slightest knowledge of how it happens.

The field of pattern recognition research is generally an endeavor to enable computers to perform the same sorts of cognitive tasks that mammals do. The secrets of the brain and consciousness have eluded current and past scientists, such that state-of-the-art pattern recognition systems can only mimic isolated cognitive capabilities of humans. For instance, accurate machine recognition of hand-writing can be performed by a computer under certain constraints, and similarly human speech can be recognised by a machine to a reasonable degree of accuracy. There is, however, no single system or methodology that can perform both tasks, even though our single brain can do all this and more.

Figure 2.1: An example scene: mirror and candle stick, M. C. Escher.

Pattern recognition has attracted interest and input from many disciplines, including mathematics, physics, engineering, computer science, psychology, biology, cognitive science, neurophysiology, statistics, geography, geophysics and operations research. The possible applications for pattern recognition are abundant in every area of industry and research. Of late, research in the area has taken the interesting turn towards nature and biology as scientists seek to model the mechanisms by which natural systems perform pattern recognition. As a result, we have areas such as artificial neural networks, evolutionary algorithms, simulated annealing, swarm algorithms and immune system algorithms.

There are several tasks that are performed under the banner of pattern recognition: detection, classification, prediction, data visualisation and analysis, and novelty detection. Pattern recognition tasks can all be assigned to one of two broad categories: *supervised* and *unsupervised*. A supervised learning process is one in which the user provides some external information about the problem. In the unsupervised case, no prior information is provided and the system is to discover the fundamental structure of the data on its own.

It would be impractical to give a comprehensive overview of the field of pattern recognition in a single chapter. Instead, this exposition discusses only those areas that are requisite knowledge for the remainder of the thesis, and focuses mainly on supervised classification.

2.3 Supervised Classification

The subject of this chapter and indeed this thesis is *supervised classification*, often referred to as *machine learning*. The task of supervised classification can be defined thus:

- A domain is given in which objects belong to one of C *classes*. The $i$the object is annotated with a *class label* $c_i \in \{1, 2, \ldots, C\}$.

- Each object is characterised by d measurements, $x_{i1}, x_{i2}, \ldots, x_{id}$. Placed in a vector, these quantities form the *measurement vector* $\mathbf{x}_i$ of object i.

- During some experimental procedure, N objects are observed resulting in a set of measurement vector-class label pairs:

$$\mathcal{D} = \{(\mathbf{x}_1, c_1), (\mathbf{x}_2, c_2), \ldots, (\mathbf{x}_N, c_N)\}$$

Each pair $(\mathbf{x}_i, c_i)$ is termed a *data sample.*

- The task is to construct a function $h(\mathbf{x})$ that learns from $\mathcal{D}$ to predict the class of a previously-unseen object. So for a new object j, we want:

  $$h(\mathbf{x}_j) = c_j$$

  The set of data samples used to learn the predictive function is called the *training set.* Each data sample in that set is called a *training sample* or *training example.*

In the case of unsupervised learning, the class labels of the data samples are not known and the algorithm must discover for itself which samples have a natural affinity with each other. Unsupervised learning will not be described further in this thesis.

The ability of a classifier to predict the class of new objects is called *generalisation.* Some examples of supervised classification are:

- A database of photographs of employees' faces is made and each photo is labeled with the respective name. A pattern recognition system is then trained on this database to identify employees as they arrive each morning, and to detect intruders who are not in the database.

- A set of pathological reports associated with patients suspected to have heart disease are catalogued over time. Each record is retrospectively labeled with the presence or absence of the disease. A pattern recognition system is then designed which learns from these samples to diagnose new patients complaining of heart trouble.

From the above examples one can see that the nature of $\mathbf{x}_i$ can vary widely, from a high-dimensional multi-spectral image (each x_{ij} corresponding to one pixel) to a set of different discrete and real quantities.

The system that achieves this goal generally consists of two stages: a *feature extraction stage* and a *classification stage.* The flow diagram is shown in Figure 2.2. The measurement vector is transformed by the pre-processor to a *feature vector* of dimensionality g, $\mathbf{y}_i = [y_{i1}, y_{i2}, \ldots, y_{ig}]$. Each element of the feature vector is called a *feature*, and is a generally non-linear transformation of the original measurements. The feature vector becomes the input to a classifier which outputs a class label c_i', the predicted class of object i. c_i' is then used in some decision process. When the predicted class does not match the true class c_i, a *misclassification* has occurred.

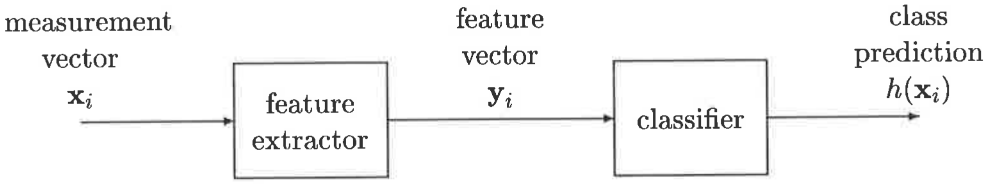


Figure 2.2: Flow diagram for a pattern recognition system.

In this very general model of supervised classification, the pre-processing stage is problem-specific while the classifier is more general. The pre-processing required for an image is quite different to that required for a time series, but the features of both could be used with a generic classifier. It is usually the classifier that learns from the data, while the pre-processor is a static entity designed by the engineer of the system. In practice, the boundary between the pre-processor and classifier is not clearly defined. For instance, artificial neural networks can be fed directly with the measurement vector, the pre-processing and classification proceeding internally and inextricably. In other instances, the pre-processor undergoes learning as well as the classifier.

This description so far has been only a basic outline; pattern classification systems can differ in many ways. An important addition to this model is the inclusion of *misclassification costs.* Consider again the above example of the use of machine classification in the diagnosis of human patients with heart disease. If the computer diagnoses a healthy patient as terminal, the worst that will happen is the patient will get a good scare and the story will appear on a situation comedy. If, however, the computer diagnoses a sick patient as healthy, the repercussions are much more grave. Therefore a cost or risk can be associated with each different type of misclassification to express its relative undesirability.

The predicted class output described previously is a *hard* decision, since it is mutually exclusive with the other cases. A classifier may also make *soft* decisions, which usually manifest in the delivery of class probabilities. Thus the predicted class label c_i' is replaced with a vector of probabilities, one for each class:

$$\mathbf{c}_i' = [p(c_i = 1|\mathbf{x}_i),\ p(c_i = 2|\mathbf{x}_i),\ \ldots,\ p(c_i = C|\mathbf{x}_C)]$$

Each $p(c_i = j|\mathbf{x}_i)$ is the *a posteriori* (or *posterior*) probability of class j given the data vector. These probabilities are usually transformed to a hard decision by selecting the class with the highest posterior probability. This approach has the advantage that a level of confidence in the final decision is provided, extremely important in those safety-critical applications.

Another paradigm for soft decisions is *fuzzy logic*, a technique for manipulating linguistic quantities in a principled manner (Ross, 1995). Each object does not belong to a single class, but rather has *membership* in a number of classes, or *fuzzy sets.* Soft decision methods have been found to be more robust to noise than their hard counterparts (Gelfand and Delp, 1991).

2.3.1 An Example: Parallelepiped Classifier

The parallelepiped classifier, commonly used in remote sensing applications (Lillesand and Kiefer, 1994), is a very simple classifier and therefore makes a good introductory example. This classifier emphasizes the view taken in the development of many non-parametric techniques: that the measurement vectors are a set of points existing in a high-dimensional space. The classifier assumes that there is only one cluster per class, and seeks to enclose each cluster in a hyper-rectangle.

The range of feature values over the data samples is calculated along each feature-space axis to determine the extents of each parallelepiped; a two-dimensional example is shown in Figure 2.3 for three classes. New samples are assigned the class label corresponding to the parallelepiped in which they fall.

A problem arises when the boxes overlap: the identity of a sample is undecided in the overlap region (for example, the points in region 1 in the figure). Similarly, the class label of a new sample that does not lie inside any of the parallelepipeds will be undecided (point 2 in the figure). These two cases of ambiguity can be resolved by assigning the class whose parallelepiped centroid is closest to the new sample.

The example of Figure 2.3 illustrates one of the main drawbacks of the parallelepiped classifier: the parallelepipeds do not account for covariance in the data, so that large regions in the corners may contain no data and may overlap with other parallelepipeds.

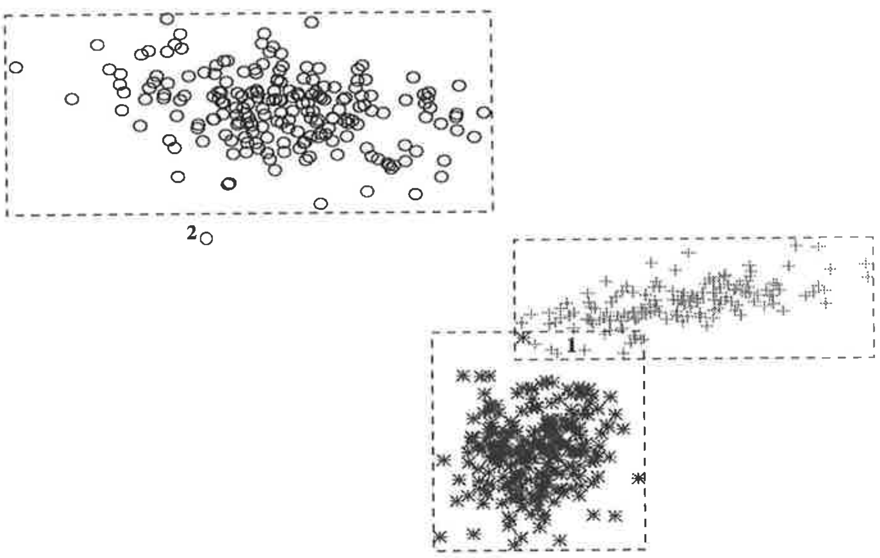


Figure 2.3: Example of the parallelepiped classifier.

2.4 Estimating Classification Error

It is a fact of life that, in real-world situations, classifiers make mistakes. The measurements chosen as inputs to the classifier are assumed to have some discriminatory qualities such that objects from the same class are similar in the measurement space. Groups of objects that are similar in the measurement space are called *clusters*. In real situations, two objects may yield the same measurement vector but belong to different classes. The two causes for this phenomenon are:

1. noise in the sensors used to obtain the measurements, and

2. overlap between clusters, such that objects from different classes share the same measurement vectors.

The issue of noise depends on the quality of the sensors used and the conditions under which the measurements are obtained. As an example, when classifying fruit based on length, the sensor may return an erroneously-large value of length for a kiwi fruit so that it is misclassified as a banana. Overlap between clusters occurs in many practical situations, and the degree of overlap is determined by the appropriate choice of measurements. For example, diameter would not be a good measurement to distinguish between apples and oranges, since objects from both classes share similar measurement values, whereas skin pigment would make an ideal discriminator. Even if the measurements are chosen carefully, class overlap still occurs due to variations in the population of all objects. For example, a particularly short banana may be mistaken for a kiwi fruit, or vice versa.

It follows that there is an irreducible classification error inherent in such situations. Therefore we take a probabilistic view of the situation via the joint distribution $P(\mathbf{x}, c)$[1] where $\mathbf{x} \in \mathcal{X}$, the universe of all objects, and c is the class of $\mathbf{x}$. Taking this perspective, there is a maximum *a posteriori*, or "most likely", value of c at a given observation which we will call $c^*(\mathbf{x})$, and consider to be the "right answer":

$$c^*(\mathbf{x}) =^{\text{argmax}}_{c} P(c|\mathbf{x})$$

[1] The capital P denotes a single probability value, while the lower-case p is a probability density function.

only for 0-1 cost

We shall see later that this is termed the *Bayes decision.*

For a fixed overall classification rule $f(\mathbf{x})$ that outputs a class prediction, the most common measure of error is the *error rate* or *probability of misclassification*, pmc :

$$\begin{aligned}
\text{pmc}\ (f) &= E_{\mathcal{X}}[\text{pmc}(\mathbf{x}|f)] \\
&= \int_{\mathbf{x}\in\mathcal{X}} p(f(\mathbf{x}) \neq c|\mathbf{x})p(\mathbf{x}).dx \\
&= \int_{\mathbf{x}\in\mathcal{X}} [1 - P(f(\mathbf{x})|\mathbf{x})]p(\mathbf{x}).dx \qquad (2.1)
\end{aligned}$$

where $E[.]$ is the expectation over the subscripted space, and $P(f(\mathbf{x})|\mathbf{x})$ is $P(c = f(\mathbf{x})|\mathbf{x})$. This is a particular case of the more general *loss* or *cost* of the classifier:

$$cost(f) = \int_{\mathbf{x}\in\mathcal{X}} L(f(\mathbf{x})|\mathbf{x}).p(\mathbf{x}).dx \qquad (2.2)$$

in which $L(f(\mathbf{x})|\mathbf{x})$ is the *expected loss* which defines the cost or loss incurred by the prediction $f(\mathbf{x})$ and subsequent action. The expectation is over the classes. In general this manifests as a *cost matrix*, whose $(i,j)th$ element $l_{i,j}$ defines the cost of predicting class i when the expected class was j:

$$L(f(\mathbf{x})|\mathbf{x}) = \sum_{j=1}^{C} l_{f(\mathbf{x}),j}P(j|\mathbf{x}) \qquad (2.3)$$

As an example, the cost matrix for the heart disease problem described earlier might be:

$$l_{i,j} = \text{diagnosis} \quad \begin{array}{r|cc|} \multicolumn{1}{r}{} & \multicolumn{2}{c}{\text{ground truth}} \\ \multicolumn{1}{r}{} & \multicolumn{1}{c}{\textit{ill}} & \multicolumn{1}{c}{\textit{healthy}} \\ \textit{ill} & 0 & 1 \\ \textit{healthy} & 1000 & 0 \end{array}$$

The misclassification rate of Equation(2.1) is the result of the commonly-used *0-1* loss:

$$l_{i,j} = \begin{cases} 0 & i = j; \\ 1 & i \neq j \end{cases} \quad i,j = 1,\ldots,C$$

The error rate resulting from 0-1 loss will be used as the primary method for measuring the performance of a classifier in this thesis. 0-1 loss is a natural measure of error for a classifier making hard decisions. For a classifier that outputs class probabilities, one possible loss function is the absolute difference between the maximum predicted probability and the predicted probability of the true class.

Although in practice we only have a single data set to go by, in theory the expected loss used in Equation(2.2) could be replaced with an expected value over training sets as well as over $P(c|\mathbf{x})$. The result is a bias-variance decomposition which has been used widely but in many different forms; some investigations are found in (Tibshirani, 1996; Wolpert, 1995; Friedman, 1996; Meir, 1994). A bias-variance decomposition can be useful to see how the expected loss is contributed to by the bias of the classifier and by its sensitivity to the data. The decomposition of (James and Hastie, 1997) is now presented, and was chosen because it is more general than the others.

In general prediction, of which classification is a special case, we have a random variable Y which we wish to predict, and another $\hat{Y}$ which is the output of our predictor. Y varies due to noise in the source[2], and $\hat{Y}$ varies with the random selection of the training set used to

[2] For classification, the source noise is the variation in the class of objects with the same measurement vector.

construct the predictor. For both variables there is a systematic component, SY and $S\hat{Y}$[3], which can be obtained by operating on the distribution of the variables (the usual choice is the expected value). The expected loss of the predictor at a specific point $\mathbf{x}$ can be written as:

$$\begin{aligned} E[L(Y,\hat{Y})] &= E[L(Y,SY)] \\ &\quad +E[L(Y,S\hat{Y}) - L(Y,SY)] \\ &\quad +E[L(Y,\hat{Y}) - L(Y,S\hat{Y})] \end{aligned} \tag{2.4}$$

where the expectation is over $P(Y|X)$ and over training sets $\mathcal{D} \in \mathcal{X}$. If the problem involves regression on a continuous variable, then squared-error loss $L_S(a,b) = (a-b)^2$ can be used, resulting in:

$$E[(Y-\hat{Y})^2] = \mathrm{var}(Y) + \mathrm{bias}^2(\hat{Y},SY) + \mathrm{var}(\hat{Y})$$

This is a familiar equation; the first term is an irreducible error due to noise, and sets a lower bound on the error. The remaining two terms form the reducible error which is a function of the classification algorithm used. The bias measures how close the predictor can get to the real function on average, while the variance indicates the sensitivity of the predictor to sampling variations in the data. Clearly we would like to reduce both terms, but decreasing the bias generally requires more sensitivity to the data, which increases the variance term. This "bias-variance trade-off" or "bias-variance dilemma" is a well-known phenomenon.

The relationship between bias and variance and their effect on overall error is not as simple for 0-1 loss. The analysis of (Friedman, 1996) revealed a counter-intuitive relationship between bias and variance, in that high bias can be canceled by low variance to improve accuracy. Several corrections to this decomposition were presented in (Wolpert, 1995) which restored the intuitive behaviour of the decomposition using covariance terms. The bias-variance decomposition for 0-1 loss can be performed under the notation used here by writing Equation(2.4) as:

$$E[L(Y,\hat{Y})] = \mathrm{var}(Y) + \mathrm{SE}(\hat{Y},Y) + \mathrm{VE}(\hat{Y},Y)$$

where the two quantities on the right are defined as:

$$\mathrm{SE}(\hat{Y},Y) = E[L(Y,S\hat{Y}) - L(Y,SY)] \tag{2.5}$$

$$\mathrm{VE}(\hat{Y},Y) = E[L(Y,\hat{Y}) - L(Y,S\hat{Y})] \tag{2.6}$$

SE is termed the *systematic effect*; it is the change in error caused by the bias $(SY - S\hat{Y})$. VE is the *variance effect*, and is the change in error caused by the variance $E[(\hat{Y} - S\hat{Y})^2]$. So rather than looking directly at the bias or variance, we examine the effect they have on the expected loss, which is of more concern.

Note that there is no unique definition of bias and variance. In (James and Hastie, 1997), variance and bias are defined as:

$$\begin{aligned} \mathrm{var}(\hat{Y}) &= E[L(\hat{Y},S\hat{Y})] \\ \mathrm{bias}(\hat{Y},SY) &= L(SY,S\hat{Y}) \end{aligned}$$

Under this definition, the bias is actually the square of the bias obtained with squared-error loss.

To simplify notation, we define:

$$\begin{aligned} P_i^Y &= P(Y=i|\mathbf{x}) \\ P_i^{\hat{Y}} &= P(\hat{Y}=i|\mathbf{x}) \end{aligned}$$

[3] The notation used here has been borrowed from (James and Hastie, 1997).

Note that $P_i^{\hat{Y}}$ involves the distribution over all possible training sets, $P(X, Y, \mathcal{D})$. For 0-1 loss the systematic parts are:

$$\begin{aligned} SY &= \overset{\text{argmax}}{i} P_i^Y \\ S\hat{Y} &= \overset{\text{argmax}}{i} P_i^{\hat{Y}} \end{aligned}$$

Let $I(.)$ be the logic function:

$$I(z) = \begin{cases} 1 & \text{if } z \text{ is True} \\ 0 & \text{if } z \text{ is False} \end{cases}$$

Using Equations(2.5) and (2.6) we get:

$$\begin{aligned} \text{VE}(\hat{Y}, Y) &= E[I(Y \neq \hat{Y}) - I(Y \neq S\hat{Y})] \\ &= P(Y \neq \hat{Y}) - P(Y \neq S\hat{Y}) \\ &= \left(1 - \sum_{i=1}^{C} P_i^Y P_i^{\hat{Y}}\right) - (1 - P_{S\hat{Y}}^Y) \\ &= P_{S\hat{Y}}^Y - \sum_{i=1}^{C} P_i^Y P_i^{\hat{Y}} \\ \text{SE}(\hat{Y}, Y) &= E[I(Y \neq S\hat{Y}) - I(Y \neq SY)] \\ &= P(Y \neq S\hat{Y}) - P(Y \neq SY) \\ &= P_{SY}^Y - P_{S\hat{Y}}^Y \\ &= \overset{\max}{i} P_i^Y - P_{S\hat{Y}}^Y \\ \text{var}(Y) &= E[I(Y \neq SY)] \\ &= P(Y \neq SY) \\ &= 1 - \overset{\max}{i} P_i^Y \end{aligned}$$

Note that while SE ≥ 0, VE can be negative with the constraint VE + SE ≥ 0. This agrees with Friedman's observation that it is possible for the variance of a classifier to cancel out its error due to bias. Now compare the variance and bias effects to the variance and bias of the classifier themselves:

$$\begin{aligned} \text{var}(\hat{Y}) &= P(\hat{Y} \neq S\hat{Y}) \\ &= 1 - \overset{\max}{i} P_i^{\hat{Y}} \\ \text{bias}(\hat{Y}, SY) &= I(S\hat{Y} \neq SY) \end{aligned}$$

There is no obvious relationship between the bias and variance and their effects on the expected loss. The decomposition is still useful for understanding how the bias and variance are affecting the accuracy of the classifier.

In general we cannot calculate pmc because we do not know $P(f(\mathbf{x})|\mathbf{x})$ or $p(\mathbf{x})$, and the domain $\mathcal{X}$ may be infinite[4]. In fact, we only have a single training set $\mathcal{D}$ from which to infer our model. Therefore any performance measures must be based on the samples provided. The experimental approach is to approximate the integral over $\mathcal{X}$ by a summation over some data set $\mathcal{T}$:

$$\widehat{\text{pmc}}_{\mathcal{T}}(f|\mathcal{D}) = \frac{1}{|\mathcal{T}|} \sum_{\mathbf{x}_i \in \mathcal{T}} I(f(\mathbf{x}_i) \neq c_i)$$

[4] Of course, there are synthetic problems where the distribution of the data is fully known.

The obvious choice for $\mathcal{T}$ is to use the data set $\mathcal{D}$ that was used to train the classifier. The resulting quantity is called the *training error.* It is widely known, however, that the training set error is a downwardly-biased estimator of pmc because the same data were used to build and to test the model. This optimistic estimate of pmc is also termed the *apparent error* (Efron and Tibshirani, 1993). In order to reliably estimate pmc , we require independent samples from $P(\mathbf{x}, c)$; such a data set is called the *test set.* Often the re-sampling of $P(\mathbf{x}, c)$ is impossible or expensive, so the supplied data is divided into a training and a test set. The classifier is built using $\mathcal{D}$ and evaluated using $\mathcal{T}$ to obtain an independent, and therefore unbiased, estimate of pmc. The test set is sometimes termed the *hold-out* sample.

The problem with the hold-out approach is that we require as much data as possible to hypothesise $f(\mathbf{x}|\mathcal{D})$, and the available data set is often not large enough that we are willing to spare samples for the test set. Furthermore, a large number of samples is required to obtain a satisfactory estimate of pmc . Consider a test set consisting of m samples, m_e of which were erroneously classified. m_e is a random variable; let us assume that it is binomially distributed:

$$m_e \sim B(m, \text{pmc})$$

The variance of our estimator $\widehat{\text{pmc}}_{\mathcal{T}}$ is:

$$\sigma^2_{\widehat{\text{pmc}}_{\mathcal{T}}} = \frac{\text{pmc} \ (1 - \text{pmc})}{m}$$

If the true error rate is 5%, say, and we want the 95% confidence interval to span no more than 2%, then we require (Ripley, 1996):

$$2\sqrt{\frac{0.05 \times 0.95}{m}} \approx 0.01$$

which yields $m \approx 1900$.

There are several re-sampling methods that can be used to improve the accuracy of the error estimate. Two are described below.

2.4.1 v-fold Cross-Validation

Cross-validation (CV) is the practice of dividing a data set into a training set $\mathcal{D}$ and a validation set $\mathcal{V}$, and subsequently using $\mathcal{D}$ to create a classifier and $\mathcal{V}$ to evaluate its performance. The difference from the hold-out method is that cross-validation is typically used for model selection, the result of which must subsequently be evaluated using the hold-out test set.

For v-fold cross-validation, the data set is randomly divided into v disjoint sub-sets of roughly equal size, $\mathcal{V}^i, i = 1, \ldots, v$. Let $\tilde{\mathcal{V}}^i$ denote the set of samples remaining when the ith cross-validation sample is removed from the whole data set. Each $\tilde{\mathcal{V}}^i$ is used to train a separate classifier $f_i(\mathbf{x})$, which is subsequently evaluated on $\mathcal{V}^i$:

$$\widehat{\text{pmc}}_{\mathcal{T}}(f_i|\tilde{\mathcal{V}}^i) = \frac{1}{|\mathcal{V}^i|} \sum_{\mathbf{x}_j \in \mathcal{V}^i} I(f(\mathbf{x}_j) \neq c_j)$$

The result is v independent estimates of error for which each sample has been used to train and to test the classifier. The overall error for the classifier is the average of these cross-validation estimates:

$$\widehat{\text{pmc}}_{\mathcal{T}}(f) = \frac{1}{v} \sum_{j=1}^{v} \widehat{\text{pmc}}_{\mathcal{T}}(f_j|\tilde{\mathcal{V}}^j)$$

The extreme case of v-fold cross-validation is *leave-one-out cross-validation,* for which $|\mathcal{V}^i| = 1, i = 1, \ldots, v$. This method results in a more accurate estimate of error, but clearly is more computationally expensive.

2.4.2 The Bootstrap

Other methods for estimating the true error rate of a classifier are obtained by adding a correction to the apparent error. One popular method is the *bootstrap* (Efron and Tibshirani, 1993), a non-parametric technique for measuring the accuracy of an estimator. It was stated earlier that the problem with estimating error is that we do not know the distribution of the data at every point. The bootstrap[5] approximates the true distribution $P(\mathbf{x}, c)$ with the empirical distribution $\hat{P}(\mathbf{x}, c)$ derived from N data vectors by placing an impulse of size $1/N$ at each $(\mathbf{x}_i, c_i)$. The empirical distribution is then re-sampled with replacement to form B bootstrap samples of size N. A non-linear estimator θ is calculated for each of the B samples yielding $\theta_1^*, \theta_2^*, \ldots, \theta_B^*$, which are averaged to obtain a more accurate estimator $\bar{\theta}^*$. The underlying assumption is that the empirical distribution is a suitable mimic for the true distribution of $\mathbf{x}$. In practice, $B = 200$ usually gives sufficiently accurate results (Efron and Tibshirani, 1993, p. 52).

The bootstrap estimate of classification error is based on the training set only, so that all of the available data can be used. To estimate classification error, two bootstrap estimates are required:

- the classification error of the model trained on the bootstrap sample and tested on the *original data*, $err(\mathbf{x}^*, \hat{P}(\mathbf{x}, c))$, and
- the classification error of the model trained on the bootstrap sample and tested on the *same bootstrap sample*, $err(\mathbf{x}^*, \hat{P}(\mathbf{x}^*, c))$.

Each of these quantities is averaged over the B bootstrap samples to obtain averages $\overline{err}(\mathbf{x}^*, \hat{F})$ and $\overline{err}(\mathbf{x}^*, \hat{F}^*)$. One might expect $\overline{err}(\mathbf{x}^*, \hat{F})$ to be a sufficient estimate of error, but in practice it is still too downwardly-biased. It is observed, however, that the error estimates obtained by testing on the bootstrap samples, the apparent errors, are on average lower than those evaluated on the original sample. We can therefore calculate the *average optimism* $\overline{err}(\mathbf{x}^*, \hat{F}) - \overline{err}(\mathbf{x}^*, \hat{F}^*)$ and add this to the apparent error of the original data set, $err(\mathbf{x}, \hat{F})$. The result is the bootstrap error estimate.

2.4.3 The Confusion Matrix

A useful tool for evaluating the performance of a classifier is the *confusion matrix*. The element $m(i, j)$ of the matrix is the number of test samples that belong to class i, and were assigned by the classifier to class j. From the matrix, one can identify which classes the classifier is confusing in its decisions. An example of the confusion matrix for a classifier used to distinguish fruit based on length is shown in Table 2.1. There are three types of fruit to be classified, bananas (class 1), apples (class 2) and oranges (class 3). When tested on 120 examples with 40 examples per class, the classifier correctly classifies all the bananas, but incorrectly classifies 9 apples as oranges, and 19 oranges as apples. The matrix can be used to see where two classes are being confused based on their sensor measurements, and new sensors can be added to particularly discriminate the confused classes. For example, based on Table 2.1 the designer might add a sensor that measures the colour or surface texture of the fruit to distinguish between the apples and oranges.

Another anomaly that can be detected with the confusion matrix is the case when the classifier tries to trade off misclassifications of one class against another. Consider, for example, the two confusion matrices for the same two-class problem shown in Table 2.2. There are 100 test samples, with 70 coming from the first class and 30 from the second.

[5] The name *bootstrap* comes from "the Adventures of Baron von Munchhausen", who was said to have pulled himself up by his bootstraps. The implication is that by using this method we are getting something for nothing.

Table 2.1: An example of a confusion matrix.

Class		predicted			Total
		1	2	3	
actual	1	40	0	0	40
	2	0	31	9	40
	3	0	19	21	40
Total		40	50	30	120

Both classifiers make 20 errors. The first classifier, Table 2.2(a), is biased towards class 1. It correctly classifies all samples from class 1, but misclassifies most of the samples from class 2. The second classifier with confusion matrix in Table 2.2(b), however, is more even-handed, and is not particularly biased towards either class. Now consider that the problem just described is heart disease diagnosis, class 1 is the class of healthy people, and class 2 is the class of people diagnosed with the disease. In this case the second classifier is more preferable than the first, because it is much more disastrous to tell a sick person that he/she is well than to tell a healthy person he/she is sick. The trade-off of errors between the classes can be controlled using a cost matrix to weight the errors.

Table 2.2: An example of a classifier that trades off errors between classes.

Class		predicted		Total
		1	2	
actual	1	70	0	70
	2	20	10	30
Total		90	10	100

(a) Classifier 1

Class		predicted		Total
		1	2	
actual	1	63	7	70
	2	2	28	30
Total		65	35	100

(b) Classifier 2

2.5 Partitioning the Data

Many classification algorithms require an independent measure of classifier performance during training, either to select from one of several models or to determine when training should stop. Therefore the following three-fold partitioning of the data is usually performed:

training set: used to train the classifier;

validation set: used to test the trained classifier in order to select a model or a stopping point; and

test set: used as a hold-out set to evaluate the generalisation performance of the final classifier resulting from the training algorithm.

The test set is a simulation of the job the classifier will be doing in the real world.

In the work presented in this thesis, the data are partitioned in contiguous chunks, displayed graphically in Figure 2.4. Let the total number of samples in the data set be N, and the actual numbers of samples in each set be n_{tr}, n_{val} and n_{tst} .

$$n_{tr} + n_{val} + n_{tst} = N$$

how do you know this decomposition is representative

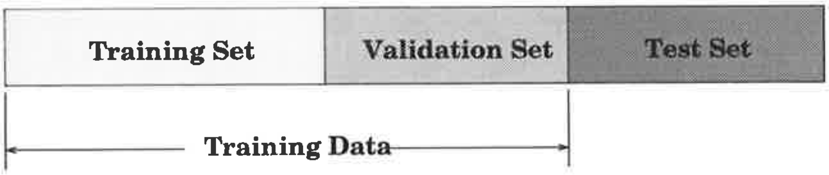


Figure 2.4: Partitioning of the data.

The training set and the validation set constitute the *training data*. **The test set is not used at all to generate the classifier**, it is only used for comparison of the generalisation performance of different classifiers. The test set estimate of percentage classification error is:

$$e = 100 \left(\frac{n_{err}}{n_{tst}} \right) \%$$

where n_{err} is the number of errors on the test set made by the classifier.

2.6 Bayesian Classification

The Bayesian approach to classification is very popular and has been for several decades. Under this approach the problem is posed in probabilistic terms and all of the probabilities and distributions are assumed to be known (Duda and Hart, 1973). The advantages of this approach are that it is theoretically well-founded, empirically well-proven and involves procedural mechanisms whereby new problems can be systematically solved (Hanson *et al.*, 1991). The pivotal mathematical tool for this analysis is *Bayes Rule*:

$$P(a|b) = \frac{P(a,b)}{P(b)}$$

Also using $P(b|a)$ we obtain the familiar form:

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

In the universe of all objects some objects are more common than others. The probability of occurrence of an object of class i is the *a priori* or *prior* probability of that class, and is denoted $P(i)$. For example, when classifying aircraft in suburban skies the prior probability of an air-liner would be much higher than that of an F-18 fighter plane. In the absence of further information, a class decision could be made based purely on the prior probabilities by always guessing the class with the highest prior probability. This static classification decision results in the *default error rate*:

$$\tilde{e} = 1 - \overset{\max}{i} P(i)$$

In practice the true prior probabilities are rarely known, and are either estimated from the data or are assumed to be equal. In the work of this thesis, the priors are assumed equal.

Rather than settle for the default error rate, we note that objects from the same class tend to have similar characteristics (that's what makes a class). So the objects from a single class are taken from some population with a probability density function (pdf) $p(\mathbf{x}|c = i)$. These conditional densities are termed *likelihood functions*, and can be modeled in different ways as discussed later.

Now consider that we have taken a measurement vector $\mathbf{x}$ of some new object whose class we do not know. The main result of Bayes decision theory is that we can invert the conditional probabilities using Bayes rule to obtain the probability of each class given the data:

$$P(c = i|\mathbf{x}) = \frac{p(\mathbf{x}|i)P(i)}{p(\mathbf{x})}$$

where $p(\mathbf{x})$ is formed by integrating over the conditional densities to form the marginal:

$$P(i|\mathbf{x}) = \frac{p(\mathbf{x}|i)P(i)}{\sum_{j=1}^{C} p(\mathbf{x}|c_j)P(c_j)} \tag{2.7}$$

This conditional class probability is called the *a posteriori* or *posterior* probability. Intuitively we would choose the class with the highest posterior probability as the class decision for $\mathbf{x}$; in this case, the denominator of Equation(2.7) is irrelevant since it is the same for all classes. Figure 2.5 shows a one-dimensional example of the likelihood functions and posterior probabilities for two normally-distributed classes with equal priors.

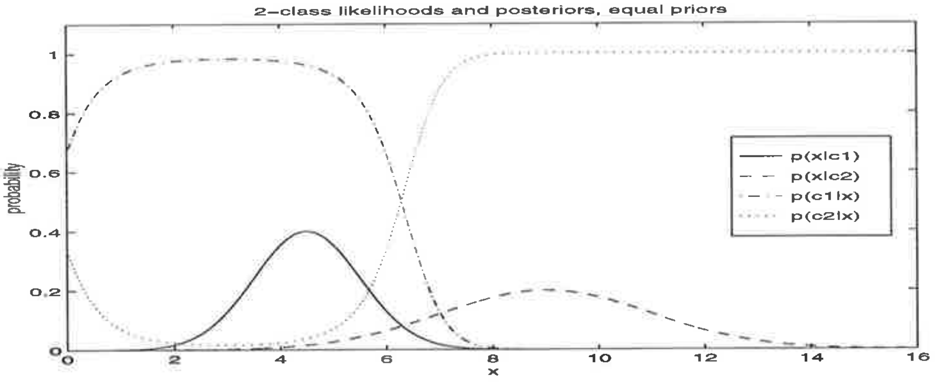


Figure 2.5: The class probability density functions and posterior probabilities plotted as a function of x for a two class problem, equal priors.

In general we must consider the loss associated with the class decision. Referring to Equations(2.2) and (2.3), the overall expected cost is minimised by minimising the decision rule's expected cost at each point $\mathbf{x}$. Re-writing the loss at a given point in the probabilistic notation:

$$L(i|\mathbf{x}) = \sum_{j=1}^{C} l(i|j)P(j|\mathbf{x})$$

The *Bayes decision rule* is to select the class c^* that minimises $L(i|\mathbf{x})$ at each point:

$$c^* =^{\text{argmin}}_{c} L(c|\mathbf{x})$$

Thus the class with the maximum posterior probability may not necessarily be selected if it has a relatively high cost.

It is an important point that the Bayes error rule results in the optimal *Bayes error rate*. This is the theoretical minimum achievable error rate of any classifier on the data; it represents the irreducible error of the problem due to overlap in the class probability density functions. The reader may be wondering "why not just use this classification rule every time and everything will be hunky dory?". The problem is that in reality we rarely know the true priors or the form and parameters of the pdfs. Therefore the performance of a classifier will only be as good as the assumptions made in deriving these quantities.

Under 0-1 loss we wish to minimise equal-cost misclassifications; the Bayes decision rule is to select the class with the maximum posterior probability:

$$c^* =^{\text{argmax}}_{c} P(c|\mathbf{x})$$

To summarise, the overall Bayesian approach for minimising error rate is to estimate or hypothesise the priors and the conditional class densities, then invert these to obtain the posteriors. For a new $\mathbf{x}$, the posterior with maximum value corresponds to the Bayes optimal class. The freedom in choice of learning algorithm is in the way the conditional pdfs are formed. Some common methods are discussed here.

2.6.1 Discriminant Functions

For the minimisation of 0-1 loss classification error, we choose the class that maximises:

$$g_i(\mathbf{x}) = p(\mathbf{x}|i)P(i)$$

$g_i(\mathbf{x})$ is called a *discriminant function*; we have C of these, and the maximum value at a point decides the class label in a winner-take-all fashion. For any two classes i and j, there is a surface at which both classes have an equal discriminant function value defined by:

$$\{x : g_i(\mathbf{x}) = g_j(\mathbf{x})\}$$

This surface is called a *decision surface* or *decision boundary*. These boundaries segment the feature space into C disjoint regions, one for each class. In general, the regions are not convex or simply-connected[6].

Since we seek a relative maximum, we can apply a monotonic increasing transformation to each discriminant function to obtain:

d. Plenty g ⟶

$$g_i(\mathbf{x}) \overset{= \log g}{=} log[p(\mathbf{x}|i)] + log[P(i)] \tag{2.8}$$

The pdfs and priors can now be plugged directly into Equation(2.8) to perform classification.

The most common practice is to assume that:

- each class consists of a single cluster;
- the samples from each class are distributed according to a multi-dimensional Gaussian function.

In that case:

$$P(\mathbf{x}|i) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}_i|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m}_i)^T\mathbf{\Sigma}_i^{-1}(\mathbf{x}-\mathbf{m}_i)}$$

where $\mathbf{m}_i$ is the mean and $\mathbf{\Sigma}_i$ is the covariance matrix of class i, d is the dimensionality of the data, and A^T denotes the transpose of matrix A. Generally the mean $\mathbf{m}_i$ and covariance matrix $\mathbf{\Sigma}_i$ of each class are estimated from the data.

The discriminant functions are:

$$g_i(\mathbf{x}) = -(d/2)\ln(2\pi) - \frac{1}{2}\ln|\mathbf{\Sigma}_i| - \frac{1}{2}(\mathbf{x}-\mathbf{m}_i)^T\mathbf{\Sigma}_i^{-1}(\mathbf{x}-\mathbf{m}_i) + \ln[P(i)]$$

Discarding the first class-independent term:

$$g_i(\mathbf{x}) = -\frac{1}{2}\ln|\mathbf{\Sigma}_i| - \frac{1}{2}(\mathbf{x}-\mathbf{m}_i)^T\mathbf{\Sigma}_i^{-1}(\mathbf{x}-\mathbf{m}_i) + \ln[P(i)]$$

[6] That is, the regions may consist of several sub-regions, or may contain holes.

2.6.2 Minimum-Distance-to-Means Classifier

With this classifier, each class distribution is assumed to have a different mean but the same covariance matrix $\mathbf{\Sigma}_i = \sigma^2 I$: that is, there are no covariance terms and the variance along each ordinate is the same for each axis for each class. The priors are also assumed equal. When the class-independent terms and scale-factors are removed, the discriminant functions become:

$$g_i(\mathbf{x}) = \mathbf{x}^T \mathbf{m}_i - \frac{1}{2}\mathbf{m}_i^T \mathbf{m}_i$$

We can obtain these functions another way. Consider the discriminant functions:

$$g_i(\mathbf{x}) = -(\mathbf{x} - \mathbf{m}_i)^T (\mathbf{x} - \mathbf{m}_i)$$

This is the negative of Euclidean distance squared from the sample to the mean of class i. Simplification reveals:

$$g_i(\mathbf{x}) = \mathbf{x}^T \mathbf{x} - 2\mathbf{x}^T \mathbf{m}_i + \mathbf{m}_i^T \mathbf{m}_i$$

which, for purposes of a comparison amongst discriminant functions, simplifies to:

$$g_i(\mathbf{x}) = \mathbf{x}^T \mathbf{m}_i - \frac{1}{2}\mathbf{m}_i^T \mathbf{m}_i$$

So the assumptions under this model are equivalent to choosing the class with the closest mean. The decision surfaces that separate the class sub-spaces are hyper-planes: each hyper-plane lies between a pair of class means and is the perpendicular bisector of the line between the means.

2.6.3 The Maximum Likelihood Classifier

The Maximum Likelihood classifier models the distribution of data in each class using a Gaussian with a different mean and a different covariance matrix. The mean and covariance matrix of each class is estimated from the training data:

$$\mathbf{m}_i = \sum_{\mathbf{x} \in C_i} \frac{\mathbf{x}}{n_i}$$

$$\mathbf{\Sigma}_i = \sum_{\mathbf{x} \in C_i} \frac{(\mathbf{x} - \mathbf{m}_i).(\mathbf{x} - \mathbf{m}_i)^T}{n_i}$$

The discriminant functions are:

$$g_i(\mathbf{x}) = -\frac{1}{2}\ln|\mathbf{\Sigma}_i| - \frac{1}{2}(\mathbf{x} - \mathbf{m}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \mathbf{m}_i) + \ln[P(i)]$$

The resulting discriminant functions between classes are quadratic; this can be very useful in exclusive-or type problems since one class can consist of several regions in feature space. An example of this classifier for synthetic two-dimensional two-class data is shown in Figure 2.6.

2.6.4 The k-Nearest Neighbours Classifier

An intuitively appealing classification rule is the *nearest neighbour* rule: find the known training sample $\mathbf{x}'$ closest to $\mathbf{x}$ and assign the class of that sample. This approach can still be considered under the Bayes framework. As the number of training samples n_{tr} approaches

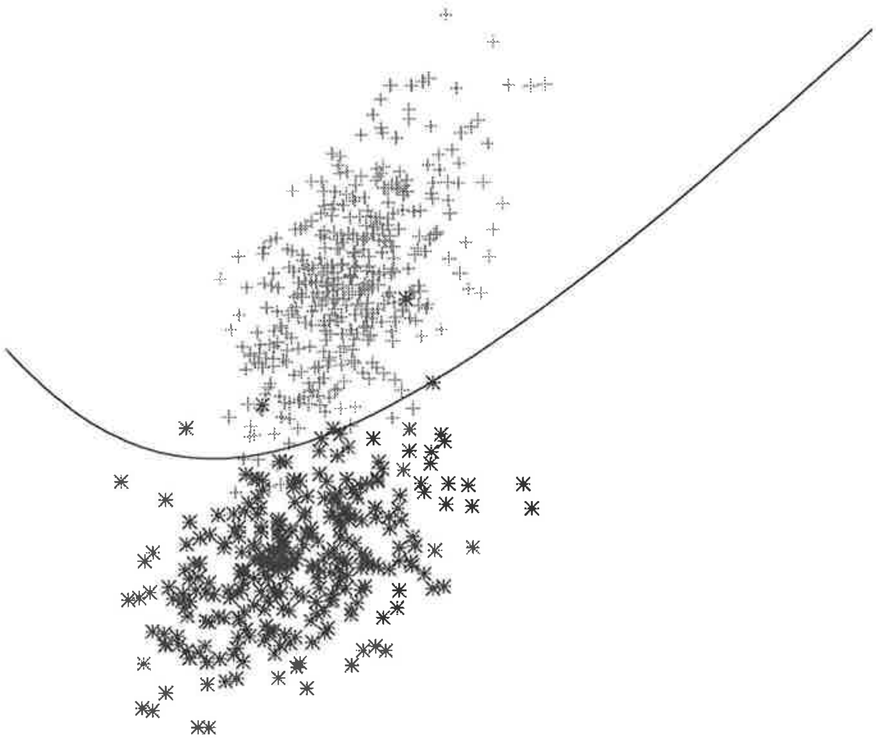


Figure 2.6: Example of the Gaussian maximum likelihood classifier.

infinity, $\mathbf{x}'$ becomes infinitely close to $\mathbf{x}$, and the class distribution at $\mathbf{x}'$ approaches the class distribution at $\mathbf{x}$:

$$\lim_{n_{tr}\to\infty} P(i|\mathbf{x}') = P(i|\mathbf{x})$$

The Bayes rule would choose the class with the maximum posterior. The nearest neighbour rule is a randomised version of the Bayes rule, in that the class label associated with $\mathbf{x}'$ has been randomly sampled from $P(\mathbf{x}, c)$. Therefore it only agrees with the Bayes rule with probability $P(c^*|\mathbf{x})$. In (Duda and Hart, 1973) it has been shown that the error rate under this rule never exceeds twice the Bayes rate.

The *k-Nearest Neighbours* (k-NN) algorithm attempts to improve upon the previous rule. It determines the k nearest training samples to the test sample being classified, and uses these samples to vote on the class label of the test sample. Ties are broken by selecting the class whose voters have the smallest total distance. The probabilistic rationale for this rule is that we now have a sampling of $P(i|\mathbf{x}')$ around $\mathbf{x}$ which is averaged through the majority vote procedure to estimate $P(c^*|\mathbf{x})$ and choose the corresponding optimal class.

In general, the higher the hyper-parameter k the more accurate the classification. There is, however, a trade-off associated with the choice of k. On the one hand, we want k to be as large as possible to improve our estimate of $P(c^*|\mathbf{x})$, but on the other hand we want the k neighbours to be as close to $\mathbf{x}$ as possible. Therefore we can expect a global minimum in error as k is varied from 1 to n_{tr}.

The decision regions created by this algorithm correspond to the Voronoi tessellation constructed from the training data. The advantages of this classifier are insensitivity to initial conditions, guaranteed convergence, and no need for training. The disadvantages are that it is slow ($\mathcal{O}(n_{tr}^2)$), the training samples must be stored with the classifier, and k must be selected.

2.7 Artificial Neural Networks

A biological *neural network*, or brain, is an interconnected conglomerate of neurons. These neurons interact with one another through *synapses*, which carry tiny electrical impulses. It is estimated that the human brain contains of the order of 10 billion neurons, each having the order of 6,000 connections to other neurons (Haykin, 1994). As a loose comparison, the pentium processor contains about 5 million transistors. Although the neurons themselves are fairly simple and operate via chemical processes that are currently well-understood, it is the extremely high level of interaction between these units that generates the complex behaviour which we can only call "ourselves".

It is the human brain that learns from and stores information that arrives at the senses. The computational paradigm of the brain is massive parallelism: the individual neurons all operate concurrently. If the neurons themselves are so limited in capability, how then do we reason and remember? The only answer can be that the information is stored not in the individual neurons, but in the patterns of interconnections between them; that is, the memory of your mother's face is not stored in a single neuron, but rather distributed among several neurons, some of which may also be partially responsible for the memory of your brother's face. Indeed, studies have revealed that brain activity is accompanied by a synchronisation in neural impulses between different parts of the brain (Gray *et al.*, 1989; Pöppel and Logothetis, 1986). The distributed nature of knowledge leads to fault tolerance, so that the loss of a few neurons does not result in the forgetting of your mother's face. Despite this distribution of information, the brain is still believed to be largely modular in structure.

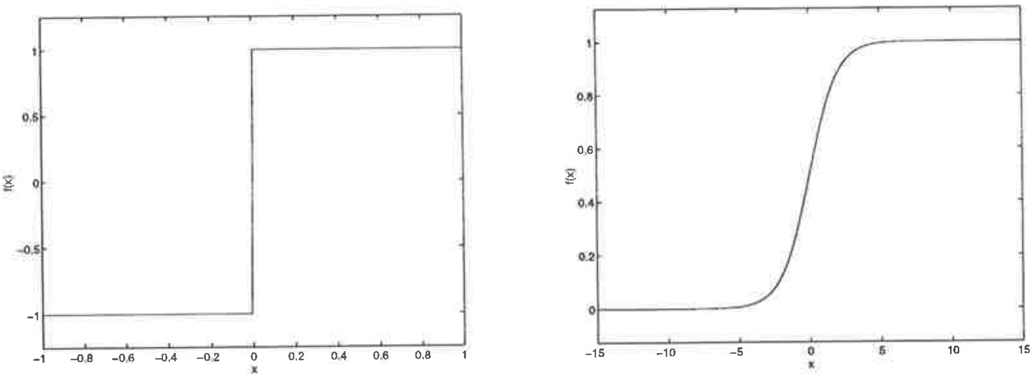
The computational paradigm of biological neural networks is quite different from digital computers which carry out their tasks in sequential fashion. *Artificial Neural Networks* (ANNs) are computational algorithms based on the virtues of the biological brain rather than on the true physical details. Interconnections of synthetic neurons are used to learn from observed data in a manner that is amenable to parallelism, robust to noise and fault tolerant. As in their biological counter-parts, the information learned is stored in the interconnections of the neurons rather than in the neurons themselves. Although the neural paradigm is parallel, ANNs are usually implemented on a sequential computer.

Artificial neurons are based on a very crude model of natural neurons (Haykin, 1994). Each neuron j receives input signals $x_1, \ldots, x_n$ from n other neurons, and sends its own output y_j to several other neurons. Each synaptic connection between neurons is weighted according to the degree to which they influence each other. w_{ji} is the numeric weight on the synapse between neuron i and neuron j. Typically the output of a neuron is a non-linear function of the weighted sum of its inputs:

$$y_j = \phi\left(\sum_{i=1}^{n} w_{ji}x_i + \theta_j\right)$$

θ_j is a bias which acts as a threshold level. The *activation function* $\phi(.)$ is typically some form of thresholding function, such as the sign function or the sigmoid function shown in Figure 2.7. The constant a in the sigmoid equation is the slope parameter; the slope at the origin is $a/4$. Learning proceeds by the iterative adjustment of the synaptic weights to optimise some objective.

Research into ANNs was first published in 1960, when the Perceptron rule and the LMS algorithm were independently discovered(Widrow and Lehr, 1990). Both methods involved the adaptation of a single neuron using an error correcting rule. Interest in ANNs deteriorated in 1969 when the famous work of Minsky and Papert revealed that the perceptron could not solve the simple exclusive-or problem (Minsky and Papert, 1969). The next development was the step to multi-layer neural networks with the MADALINE, which used a hard-limiting



$$\phi(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$

$$\phi(x) = \frac{1}{1+e^{-ax}}$$

$$\phi'(x) = a.\phi(x).[1 - \phi(x)]$$

Sign function

Sigmoid function

Figure 2.7: Neuron activation functions.

1974?

activation function. This network structure was able to solve the exclusive-or problem, but could not be trained. Although a multi-layer training algorithm was discovered soon after by Werbos in 1971, it was not popularised until 1985 when it was re-discovered by Parker, Rumelhart, Hinton and Williams. The logical step that took nearly 16 years to gain popular usage was to use continuous, differentiable activation functions which approximate the thresholding function.

There are several types of ANN which are fundamentally different in structure and purpose. Of these, some are used for supervised learning and others for unsupervised learning. The supervised networks can be divided into two types: those that learn via gradient descent and those that learn via an error-correcting rule (Widrow and Lehr, 1990). An example of each type is described below. Both types of learning operate on the *minimal disturbance principle*, which suggests that new information be learned in such a way as to cause minimal disturbance to information previously learned (Widrow and Lehr, 1990). These networks are termed *feed-forward* networks, because they consist of several layers of neurons with the output of each neuron in layer k connected to the input of each neuron in layer $k + 1$. Data is presented at the input layer and the signals propagate through the layers of the network to produce a result at the output layer. Such networks can be used for classification and, more generally, function approximation.

2.7.1 The Generalised Linear Machine

The earliest neural elements, the Adaline and the Perceptron, were both trained with an error-correcting rule and used a discrete activation function. These algorithms are useful for classification, since the output yields a hard yes-no decision. The visual interpretation of the Perceptron is a hyper-plane that divides the input space into two, providing a dichotomy of the data points in that space. An example for points in two dimensions is shown in Figure 2.8. A point is assigned to one class or the other based on the magnitude of its dot product with the weight vector.

This dichotomiser is generalised to a multi-class situation by the Generalised Linear Machine (GLIM) classifier (Nilsson, 1993). The GLIM consists of a single-layer of winner-

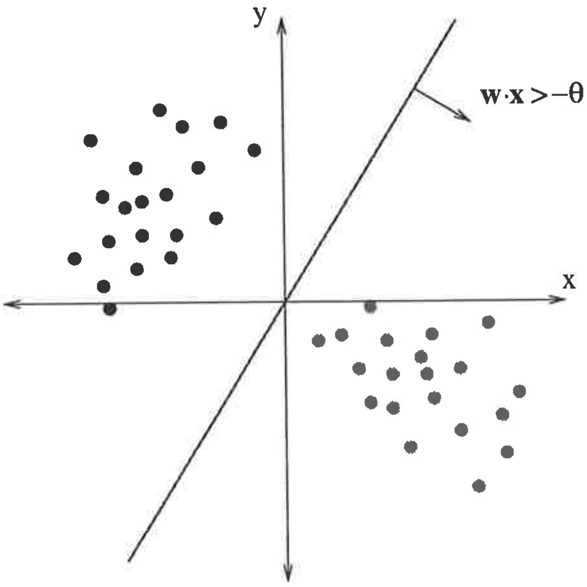


Figure 2.8: An example of a 2-dimensional hyper-plane separating two classes.

take-all neurons, one for each of the C unique classes. The classifier is shown in Figure 2.9. The term "winner-take-all" comes from competitive learning: the neuron having the largest output is chosen as the winner of the competition, and its weights alone are updated. The d-dimensional input vector, $\mathbf{x} = [x_1, x_2, \ldots, x_d]$, is fed as input to the classifier.

At the ith node, the discriminant function for the ith class $g_i(\mathbf{x})$ is:

$$g_i(\mathbf{x}) = \sum_{j=1}^{d} x_j w_{ij} + \theta_i$$

To unify the neuron bias with this notation, we augment the input vector with a constant input of 1:

$$\mathbf{x}' = [1, x_1, x_2, \ldots, x_d]^T$$

and set the bias to be the zero-th weight $\mathbf{w}_{i0}$. We can then write the discriminant function as the dot product of the augmented input with the node's $(d+1)$-dimensional weight vector $\mathbf{w}_i$:

$$g_i(\mathbf{x}) = \mathbf{w}^T\mathbf{x}'$$

The predicted class of the input vector, c', corresponds to the node with the maximum output:

$$c' = \overset{\text{argmax}}{i}\ g_i(\mathbf{x})$$

Training proceeds as follows. Let the weight vector at the nth iteration be denoted $\mathbf{w}_i(n)$. The weight vectors are initialised to zero; $\mathbf{w}_i(0) = 0$. For each training vector $\mathbf{x}$ with correct class label c, the predicted class c' is calculated. If $c' = c$, the weights are updated according to the following equation:

$$\mathbf{w}_c(n+1) = \mathbf{w}_c(n)$$

If the predicted class is wrong, however, then $c' \neq c$ and the weight vectors are updated according to:

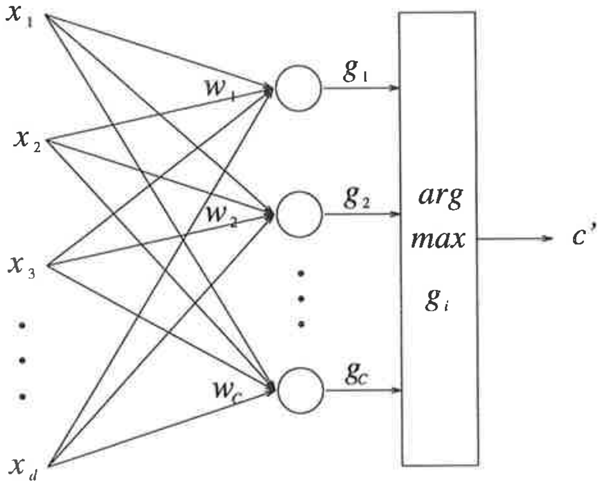


Figure 2.9: The Generalised Linear Machine.

$$\begin{aligned}\mathbf{w}_c(n+1) &= \mathbf{w}_c(n) + \mu\mathbf{x}' \\ \mathbf{w}_{c'}(n+1) &= \mathbf{w}_{c'}(n) - \mu\mathbf{x}'\end{aligned}$$

where μ is the learning rate. Thus the weights are modified so as to correct the error most recently made.

When each training vector has been presented once, an *epoch* has transpired. Training continues until a prescribed error rate is achieved, or a prescribed maximum number of epochs has occurred. If the epoch limit is reached, the classifier did not converge, probably because the data are not linearly separable. The subsequently-used weights are taken from the epoch that resulted in the least training error. For linearly-separable data and $\mu > 0$, convergence is guaranteed (Nilsson, 1993, pp. 88-90).

The advantages of the GLIM are its simplicity and speed ($O(N.e_{av})$, where e_{av} is the average number of epochs for training), and its insensitivity to the choice of initial weights and learning rate. The disadvantages are its inflexible linear decision surfaces and uncertain time to convergence. Note that although a linear classifier has simple decision surfaces, non-linearly transforming the data before classification is equivalent to using more complex decision surfaces with the original data.

2.7.2 Multi-Layer Perceptron

The *multi-layer perceptron* (MLP) is a feed-forward network with an arbitrary number of layers. The general architecture is shown in Figure 2.10. The network is organised into layers with an input layer, an output layer, and hidden layers in between. Usually either one or two hidden layers are used. Connections traditionally only exist between adjacent layers, although short-cut connections can be added between a neuron and neurons in arbitrary successive layers.

The activation function at each hidden neuron is typically the sigmoid function shown in Figure 2.7; in practice any differentiable function can be used. The activation functions at the output neurons can be either linear (*ie:* $\phi(a) = a$) or sigmoidal. The MLP is trained using a set of input-output data pairs $\{\mathbf{x}_i, \mathbf{d}_i\}^N$. The objective of training is to learn the required mapping from the input domain to the output domain. Training proceeds by the iterative presentation of input data vectors and subsequent adjustment of the weight vectors. Let $\mathbf{x}(n)$ be the input vector presented to the input layer of the network at iteration n. These

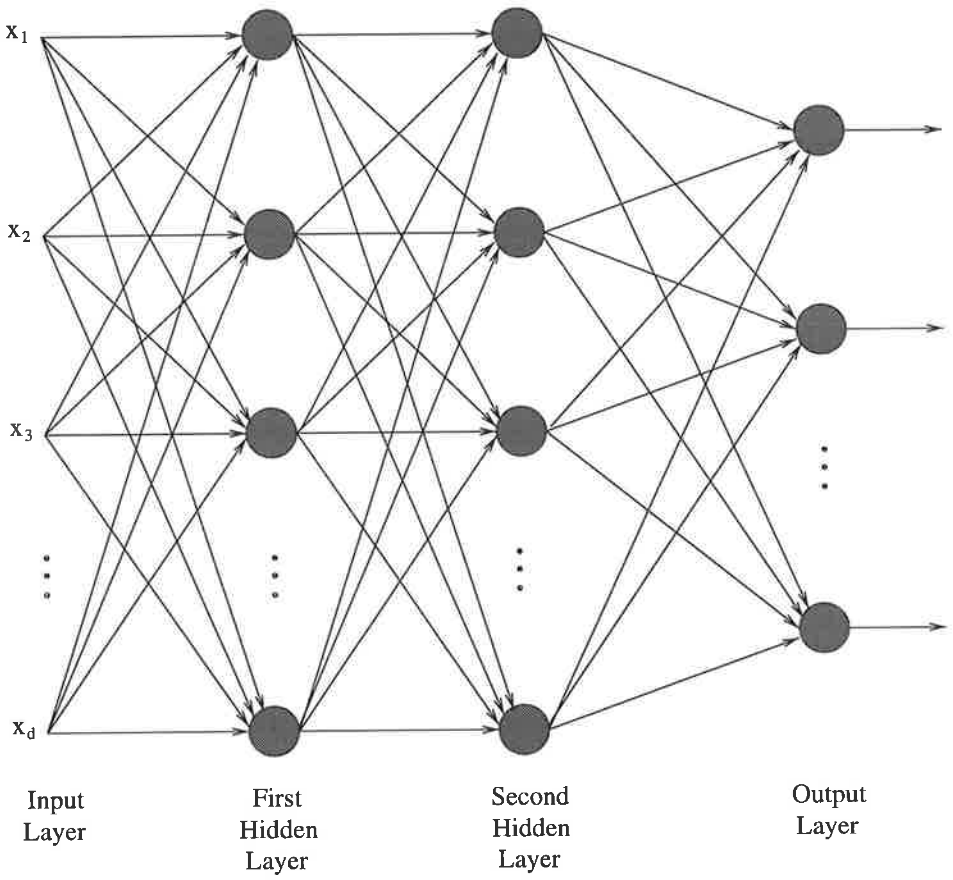


Figure 2.10: The Multi-Layer Perceptron architecture.

signals are propagated forward through the network to arrive at the output neurons, thus resulting in an output vector.

We employ the following notation: $y_j^k(n)$ is the output of the jth neuron in layer k as a result of the presentation of a training sample at iteration n. The vector of outputs of the n_k neurons contained in layer k is $\mathbf{y}^k(n)$. Each neuron j (except for the input-layer neurons) has an associated weight vector $\mathbf{w}_j^k(n)$, where n is the current iteration. The synapse connecting the output of neuron i in layer $k-1$ to the input of neuron j in layer k has the weight $w_{ji}^k(n)$. Each layer except the output layer is prepended with a neuron that has no inputs, and constantly outputs the value +1 (*ie:* $y_0^k(n) = 0 \; \forall \; n$). This allows the graceful inclusion of the bias term $w_{j0}^k(n) = \theta_j^k$ into the notation.

The input layer is labelled layer 0, and contains one node for each of the input data dimensions. Neurons in this layer have no weights or activation function: their outputs are simply the corresponding input data vector elements:

$$\mathbf{y}_i^0(n) = \mathbf{x}(n)$$

For subsequent layers, the neuron outputs are the weighted sums of the outputs of the previous layer acted upon by the activation function:

$$\begin{aligned} y_j^k(n) &= \phi_k(\mathbf{w}_j^k(n)^T.\mathbf{y}^{k-1}(n)) \\ &= \phi_k\left(\sum_{i=1}^{n_{k-1}} w_{ji}^k(n) y_i^{k-1}(n) + w_{j0}^k(n)\right); \quad k = 1, \ldots, K \end{aligned}$$

where $\phi_k(.)$ is the activation function used at layer k. Eventually the output vector is obtained from the network, $\mathbf{y}^K(n)$. The error vector at the output is the difference between the desired and network output vectors:

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{y}^K(n) \tag{2.9}$$

The goal of training is to minimise some function of this error vector, usually the *instantaneous sum-squared error*, which is the sum-squared error at the outputs upon presentation of the training sample at iteration n:

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j=1}^{n_K} e_j^2(n) \tag{2.10}$$

We really want to minimise the *average sum-squared error* over the set of training samples:

$$\mathcal{E}_{av} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{E}(n) \tag{2.11}$$

The free parameters of the network, the synaptic weights, must be modified to minimise this average error. The weight vectors are randomly initialised prior to training. The gradient descent algorithm used to adjust the weights is called *error back-propagation*. While the error function to differentiate is the average error in Equation(2.11), in practice the instantaneous error of Equation(2.10) is often used to approximate the true gradient. The adjustment $\Delta w_{ji}^k(n)$ to weight $w_{ji}^k(n)$ is formed using this gradient, and the weight is then updated according to the *delta rule*:

$$w_{ji}^k(n+1) = w_{ji}^k(n) + \Delta w_{ji}^k(n)$$

with:

$$\Delta w_{ji}^k(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}^k(n)}$$

where η is the *learning rate* parameter. Define the *internal activity* of neuron j from layer k as:

$$y_j^k(n) = \mathbf{w}_j^k(n)^T.\mathbf{y}^{k-1}(n))$$

The gradient of the instantaneous error with respect to weight w_{ji}^k is obtained using the chain rule:

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}^k(n)} = \frac{\partial \mathcal{E}(n)}{\partial y_j^k(n)} \frac{\partial y_j^k(n)}{\partial v_j^k(n)} \frac{\partial v_j^k(n)}{\partial w_{ji}^k(n)}$$

For all layers, the last two partial derivatives are:

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \phi_k'(v_j^k(n))$$

and:

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i^{k-1}(n)$$

The other term depends on whether the neuron is in the output layer or in a hidden layer. We shall call this partial derivative the *local gradient* at neuron j. For the output neurons:

$$\frac{\partial \mathcal{E}(n)}{\partial y_j^K(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j^K(n)}$$

Referring to Equations(2.9) and (2.10):

$$\frac{\partial \mathcal{E}(n)}{\partial y_j^K(n)} = -e_j(n)$$

The situation is more involved for neurons in the hidden layer, since there is no desired output with which to compare the neuron's output. We exploit the fact that we know this partial derivative for the neurons in the next layer, hence the term error back-propagation. If we consider the instantaneous error $\mathcal{E}(n)$ to be a multivariate function of the outputs in layer $k+1$, then using the chain rule for multivariate functions:

$$\begin{aligned}\frac{\partial \mathcal{E}(n)}{\partial y_j^k(n)} &= \sum_{i=1}^{n_{k+1}} \frac{\partial \mathcal{E}(n)}{\partial y_i^{k+1}(n)} \frac{\partial y_i^{k+1}(n)}{\partial y_j^k(n)} \\ &= \sum_{i=1}^{n_{k+1}} \frac{\partial \mathcal{E}(n)}{\partial y_i^{k+1}(n)} \phi_{k+1}'(v_i^{k+1}(n)).w_{ij}^{k+1}(n); \quad k = 1, \ldots, (K-1)\end{aligned}$$

Thus the local gradient can be computed in terms of the local gradients at the nodes in the succeeding layer.

When each training sample has been presented to the network once, an *epoch* has occurred. Training continues until some criterion is reached, such as zero training set error, an increase in validation set error or the gradient magnitude approaches zero. Note that the weight updates are different if we calculate the gradient with respect to the average error:

$$\frac{\partial \mathcal{E}_{av}(n)}{\partial w_{ji}^k(n)} = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1} n_K e_k(n) \frac{\partial e_k(n)}{\partial w_{ji}^k(n)}$$

This is called the *batch* method of training, as opposed to the *pattern* method that was described earlier. Although the gradient calculated using the average error is more accurate, the pattern method requires less storage in implementation and results in a stochastic estimate of the gradient, which can help in escaping local optima (Haykin, 1994). In general, the efficiency of each approach depends on the problem.

What sort of functions can the MLP approximate? The *universal approximation theorem* addresses this question:

not Haykin

Theorem 2.1 (Universal Approximation Theorem (Haykin, 1994)) *Let $\phi(.)$ be a non-constant, bounded, and monotone-increasing continuous function. Let I_p denote the $p-$dimensional unit hypercube $[0,1]^p$. The space of continuous functions on I_p is denoted by $C(I_p)$. Then given any function $f \in C(I_p)$ and $\epsilon > 0$, there exist an integer M and sets of real constants α_i, θ_i and w_{ij}, where $i = 1, \ldots, M$ and $j = 1, \ldots p$ such that we may define:*

$$F(x_1, \ldots, x_p) = \sum_{i=1}^{M} \alpha_i \phi \left(\sum_{j=1}^{p} w_{ij} x_j - \theta_i \right)$$

as an approximate realisation of the function $f(.)$; that is,

$$|F(x_1, \ldots, x_p) - f(x_1, \ldots, x_p)| < \epsilon$$

for all $\{x_1, \ldots, x_p\} \in I_p$.

The reader should note well that this is an existence theorem, and does not say what the optimal number of hidden nodes or weight values are. In addition, the theorem states that only a single hidden layer is required to approximate an arbitrary continuous function, whereas the use of multiple hidden layers may greatly decrease the size or training time required for the network. This theorem has been generalised to allow the use of any smooth (thrice differentiable) non-linear activation function $\phi(.)$ in (Kreinovich, 1991).

The MLP can be applied to classification problems, since these require a mapping from the input domain to the class decisions. The most common architecture is to use one output-layer node per class and construct the target vectors using a 1-of-m encoding:

$$\begin{aligned} \mathbf{d}^T(n) &= [0, 0, \ldots, 1, \ldots, 0] \\ \text{with} \quad d_i(n) &= \begin{cases} 1 & \text{if } i = \text{ class of } \mathbf{x}(n) \\ 0 & \text{otherwise} \end{cases} \quad ; i = 1, \ldots, C \end{aligned}$$

not original ref

It has been shown that in the case of infinite training data, the network with globally-optimal weights approximates the class posterior probabilities at the output neurons (Haykin, 1994):

$$y_j^K(n) = P(j|\mathbf{x}(n)); \quad j = 1, \ldots, C$$

Thus the optimal class decision is the class corresponding to the output-layer neuron with the largest output. Note that under this encoding, minimisation of the sum-squared-error at the output is not the same as the minimisation of misclassification error, which would be a discontinuous function.

2.7.2.1 Practical Issues

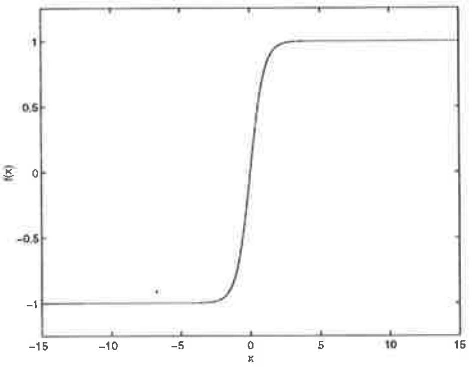
There are several practical considerations that must be addressed to successfully use the MLP.

Architecture The first decision that must be made is the choice of architecture. If too many hidden neurons are used, the network will tend to over-fit the data; if too few, the network may under-fit since it does not possess enough free parameters to perform the mapping. In addition, the number of layers may make a difference to the performance of the MLP. We could go as far as to allow arbitrary forward connections among the neurons. These factors depend on the underlying complexity of the data which is generally unknown, so for near-optimal results a search must be made over the set of possible architectures.

There is a significant amount of research into the use of evolutionary algorithms for this purpose, the different methods varying in the extent of the search (Branke, 1995).

For instance, genetic algorithms have been used to find the best neural connections for a fixed number of hidden nodes (Miller *et al.*, 1989), while more extreme approaches have used genetic programming to find the number of hidden layers, hidden nodes, the interconnections between them and the initial weights for training (Zhang and Mühlenbein, 1995).

Activation Functions It has been found that the training performance of an MLP can be improved by using an anti-symmetric activation function at the hidden nodes (Haykin, 1994; Bishop, 1995). The usual choice is the *hyperbolic tangent* shown in Figure 2.11.



$\phi(x) = \frac{1-e^{-ax}}{1+e^{-ax}}$

Figure 2.11: Hyperbolic tangent activation function.

Training can also be expedited for classification by modifying the target vector representation. With the binary target values set to the asymptotic values of the activation function, the weights can become very large and training become slow. This situation can be remedied by offsetting the target values by a small amount ϵ:

$$\mathbf{d}^T(n) = [\epsilon,\ \epsilon,\ \ldots,\ (1-\epsilon),\ \ldots,\ \epsilon]$$

Local Optima In practice it is highly unlikely that the error function will contain a single optimum; rather the function that is being optimised will tend to be very rough and noisy, and dotted with local optima (Widrow and Lehr, 1990). Since back-propagation training is a gradient descent method, convergence to a local optimum is final and the network becomes trapped at this sub-optimal point. Therefore two networks with the same architecture can often achieve the same goal using different sets of weights. It is sensible to train the network several times, starting with different initial weights for each training instance. For each training run, the weight vector may start in a different basin of attraction, and hopefully the basin of the global optimum will eventually be encountered.

One method for escaping small local optima is the use of a *momentum* term in training, although this adaptive step-size update method was explicitly developed to speed up training. The delta rule is modified to include a memory of previous step sizes (Haykin, 1994):

$$\Delta w_{ji}^k(n) = \alpha \Delta w_{ji}^k(n-1) - \eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}^k(n)}$$

where α is the momentum term, and $0 \leq |\alpha| < 1$ for stability. Momentum allows the step-size parameter to adapt to the local landscape: in regions where successive updates involve gradients with the same sign, the weight update increases in magnitude, metaphorically gaining momentum. In the case where gradients change sign on successive updates, the weight vector has overshot a local minimum and the weight update magnitude decreases to fit in the smaller basin. For an appropriately chosen α, the weight vector can overshoot and thus escape relatively small local optima.

Generalisation Generalisation is a significant issue for MLPs: one could keep adding hidden nodes to a network and generally the training error would reduce, perhaps reaching a perfect classification accuracy. However, the network would probably be totally unable to generalise since it would be tuned specifically to the data. Thus the very objective function we minimise, the training set error, leads to the wrong network!

A useful result has been published by Baum and Haussler for an MLP with one hidden layer used as a binary classifier (Baum and Haussler, 1989). Given a fixed architecture with M hidden neurons and W synaptic weights, the network will almost definitely generalise well, provided that:

1. the fraction of errors made on the training set is less than $\epsilon/2$;

2. the number of training samples N satisfies:

$$N \geq \frac{32W}{\epsilon} \ln \left(\frac{32M}{\epsilon} \right) \tag{2.12}$$

where ϵ is the fraction of errors permitted during test. This is a *distribution-free, worst-case* formula; in practice a much smaller training set may be satisfactory. To a first approximation, Equation(2.12) reduces to the rule-of-thumb:

$$N > \frac{W}{\epsilon}$$

This result assumes that we have the number of hidden nodes most appropriate for the complexity of the problem. In practice one tends to have a fixed-size training set and wishes to find the best architecture. To avoid over-fitting for networks that are too large, cross-validation techniques can be used to stop training when generalisation is estimated to be at its best.

Pre-Processing Experience has shown that transforming the data before use with an MLP can improve performance (Bishop, 1995). The specific transformation depends on the type of data:

real-valued: the input variable x is typically de-meaned and scaled to the same dynamic range as the activation functions used. This allows the same range to be used for the random initialisation of all weights, regardless of the average magnitude of the original data measurements.

enumerated: if x can only assume one of m possible values, and these values have no natural ordering, x can be encoded in one-of-m form to ensure that no artificial ordering is imposed. For example if $x \in \{red, green, blue, yellow\}$ then x could be replaced with four binary inputs:

x_1	x_2	x_3	x_4	x
1	0	0	0	red
0	1	0	0	green
0	0	1	0	blue
0	0	0	1	yellow

Enumerated variables are also referred to as *nominal* or *categorical* attributes.

2.7.2.2 RPROP Training Algorithm

The results of back-propagation training are sensitive to the weight step-size η, the correct choice of which is problem-dependent. The use of momentum may improve matters, but in practice moves the selection of an important parameter back one level. One can imagine also that the learning rate should have a different magnitude at different stages in training. Several adaptive learning-rate techniques, such as the Delta-Bar-Delta technique, have been developed to modify the learning rate in response to the behaviour of the error function. Unfortunately the step-size is often modified independently of the *magnitude* of the partial derivative $\frac{\partial \mathcal{E}(n)}{\partial \omega_{ij}}$, and learning is sensitive to the unforeseen behaviour of this derivative.

RPROP (*Resilient PROPagation*) (Riedmiller and Braun, 1993) is an adaptive weight update method in which each weight step is based on only the sign of the local gradient, and not the magnitude. Ignoring the magnitude of the local gradient results in more robust behaviour with respect to learning rate parameters. Each weight has an individual update value Δ_{ij}, which solely determines the size of the weight update:

$$\Delta_{ij}^{(n)} = \begin{cases} \eta^{+} \times \Delta_{ij}^{(n-1)} & \text{if } \frac{\partial E}{\partial \omega_{ij}}^{(n-1)} \times \frac{\partial E}{\partial \omega_{ij}}^{(n)} > 0, \\ \eta^{-} \times \Delta_{ij}^{(n-1)} & \text{if } \frac{\partial E}{\partial \omega_{ij}}^{(n-1)} \times \frac{\partial E}{\partial \omega_{ij}}^{(n)} < 0, \\ \Delta_{ij}^{(n-1)} & \text{otherwise.} \end{cases}$$

where $0 < \eta^{-} < 1 < \eta^{+}$, and the local gradient $\frac{\partial E}{\partial \omega_{ij}}^{(n)}$ is calculated at epoch n. Verbally, the weight update-value is increased when the derivative keeps the same sign so as to speed up convergence in shallow regions of the error surface. If the gradient changes sign, the network has over-shot a local minimum in weight-space and the update-value is decreased in magnitude accordingly.

With the local update-value calculated, the weight-update itself is:

$$\Delta\omega_{ij}^{(n)} = \begin{cases} -\Delta_{ij}^{(n)} & \text{if } \frac{\partial E}{\partial \omega_{ij}}^{(n)} > 0, \\ +\Delta_{ij}^{(n)} & \text{if } \frac{\partial E}{\partial \omega_{ij}}^{(n)} < 0, \\ 0 & \text{otherwise.} \end{cases}$$

$$\omega_{ij}^{(n+1)} = \omega_{ij}^{(n)} + \Delta\omega_{ij}^{(n)} \tag{2.13}$$

The weights are updated only after an epoch (*ie:* batch training). The exception to Equation(2.13) occurs when the partial derivative changes sign, *ie:* the previous step was too large. In this case, the weight is reverted to its previous value:

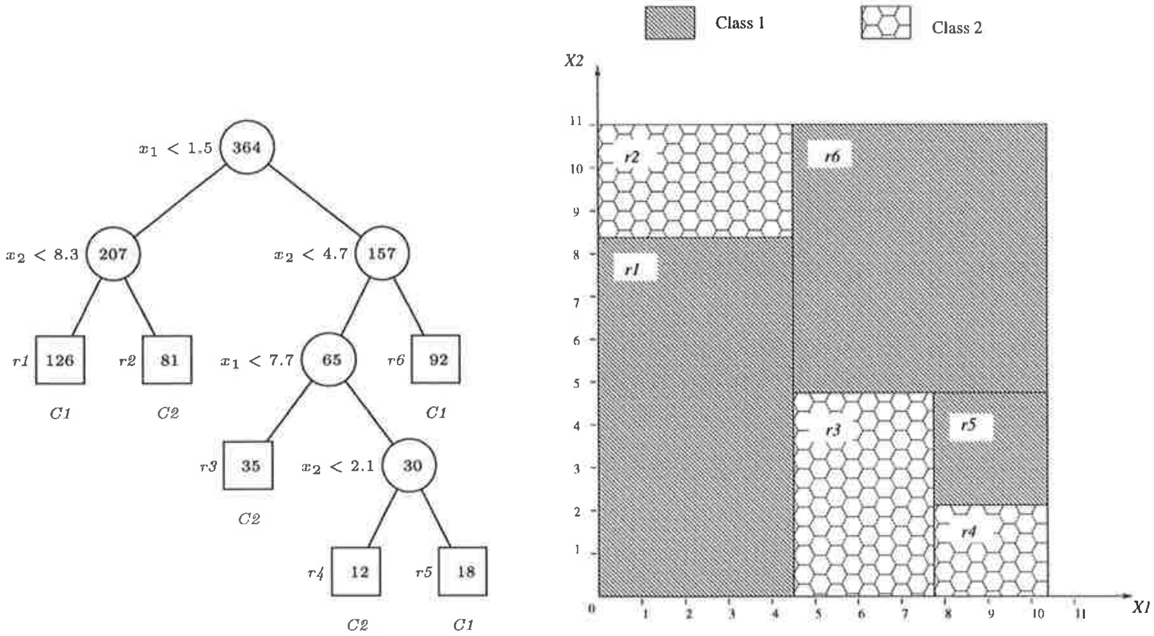
$$\Delta\omega_{ij}^{(n)} = -\Delta\omega_{ij}^{(n-1)} \quad \text{if } \frac{\partial E}{\partial \omega_{ij}}^{(n-1)} \times \frac{\partial E}{\partial \omega_{ij}}^{(n)} < 0$$

When back-tracking is performed, the gradient changes sign again. To avoid back-tracking repeatedly, the gradient $\frac{\partial E}{\partial \omega_{ij}}^{(n-1)}$ is set to zero for the next weight update. The update-values Δ_{ij} are initialised randomly with a uniform distribution on $[\Delta_{min}(0), \Delta_{max}(0)]$. During training, the update-values are constrained to the range $[\Delta_{min}, \Delta_{max}]$. The weights are initialised randomly on the range $[\omega_{min}, \omega_{max}]$ with a uniform distribution.

2.8 Decision Trees

Decision trees are data models that can be used to classify objects by asking a series of questions to arrive at a class prediction. The trees, which are generally binary, consist of internal nodes and terminal nodes; an example is shown for a 2-dimensional domain with two classes in Figure 2.12(a). Each node represents a sub-set of the training data: at the root node we begin with the full training set, which is subdivided as we progress down the tree. A question is asked at each internal node, and all the samples responding affirmatively are assigned to the left child node, with those responding negatively going to the right child node. This process is called a *split*. Usually splits involve one variable and are of the form "Is $x_i < c$?". In the example, we begin with 364 training examples. For 207 of these samples, the first attribute x_1 is less than 1.5, and these samples are assigned to the left branch. The subdivision of the data sample continues until a terminal node is reached. Each terminal node is assigned one of the pre-existing class labels according to the construction algorithm. Thus a novel sample presented to the tree can successively be tested against the splitting questions along some path and eventually arrive at a terminal node where a class label is assigned.

Referring to Figure 2.12(b), we can see that each region r_i corresponds to one of the terminal nodes in Figure 2.12(a). The axis-parallel sub-division of the feature space is a characteristic of this form of univariate split.



(a) Decision Tree for 2-class 2-dimensional problem. The number of samples associated with each node is written in the node, the splitting criterion is shown to the left of each internal node, and the assigned class number is written underneath each terminal node.

(b) Corresponding partition in 2-dimensional domain.

Figure 2.12: An example of a decision tree.

The decision tree approach to classification has the advantage of being descriptive as well as predictive, since the rules of a reasonably-sized tree are humanly interpreted and understandable. The three main issues in the construction of a decision tree are (Breiman *et al.*, 1984):

1. selection of the splits,

2. when to stop splitting, and

3. the assignment of each terminal node to a class.

There are very many ways of addressing these issues, since the research field is quite mature and many different algorithms have been developed. Among the better-known algorithms are CART (Breiman *et al.*, 1984), ID3 (Winston, 1992), C4.5 (Quinlan, 1993), FOIL (Quinlan and Cameron-Jones, 1995), FACT (Loh and Vanichsetakul, 1988) and QUEST (Loh and Shih, 1997*b*). A description of CART, C4.5 and QUEST are included in this section. A more comprehensive overview of the whole field can be found in (Murthy, 1996; Gelfand and Delp, 1991). But first, a discussion of some of the main points about decision trees.

One can observe from the previous example that univariate splits can be quite inappropriate for data in which class discrimination depends on combinations of variables. While early endeavours in decision tree research only used univariate splits, multi-variate splits are now commonplace, and usually manifest as linear (oblique) splits of the form (Breiman *et al.*, 1984):

$$\sum_m a_m x_m \leq c\ ?$$

In general, arbitrary combinations of the data can be used to perform splits. For instance, neural networks have been used to perform non-linear multivariate splits at each internal node (Guo and Gelfand, 1992).

Another advantage of decision trees is their ability to cope with ordered and categorical data. Often a variable can only take on one of several values which have no particular ordering, such as colours or types of car. These enumerated variables can pose a problem for some methods. Given a variable x_i which can take on values in $A = \{a_1, a_2, \dots a_K\}$, the split question for an enumerated variable is:

$$x_i \in B, \text{where} B \subseteq A$$

Decision trees also allow for a rather unique method of dealing with missing data, called *surrogate splits* (Breiman *et al.*, 1984). A measure of similarity of two splits is defined for a given node. If the best split S is performed on variable x_m, then find the split S' that uses variables other than x_m and that is most similar to S. S' is the best surrogate split. Now find the second best surrogate split, the third best and so on. When it comes to using the tree for classification, a sample that has the value of x_m missing is tested on the best surrogate split, and so on for any other missing values.

A drawback of decision trees is that they are grown incrementally according to some criterion, and are therefore incrementally optimal rather than globally optimal. The use of hard decisions may also be a liability in noisy environment, due to their sensitivity to noise. Developments in decision trees have been made using soft decisions, and fuzzy logic has also been used to partition data. Soft splitting assigns probabilities to sample categorisations, so that a sample has a membership in multiple tree nodes. This can result in more robustly interpretable trees (Gelfand and Delp, 1991; Sethi, 1995).

Although decision trees can be understood, they must be interpreted with caution. For example, if a variable is not used to perform a split in the tree, we might conclude that it is not useful for classification. It may be, however, that the variable's usefulness was masked by other variables (Breiman *et al.*, 1984). When this masking is narrow, small changes in the data sample or the priors may greatly affect the final tree, leading to unstable results. There are methods to cope with tree interpretation described in (Breiman *et al.*, 1984), but the most sage advice from there is to do any such analysis carefully and objectively.

Algorithms that split on variables by performing an exhaustive search have a computational complexity that increases with the number of values taken by that variable in the learning sample. For an ordered variable with n distinct values, there are $(n-1)$ splits, and for an enumerated variable with M values, there are $(2^{M-1}-1)$ splits (Loh and Shih, 1997*b*). A problem that affects tree interpretability is that exhaustive search techniques tend to be biased towards variables that have more splits.

2.8.1 CART

The book on *Classification and Regression Trees* (Breiman *et al.*, 1984) (CART) is a seminal work on decision trees which discusses and offers solutions for many of the issues in the field, including generalisation and parsimony. It was observed in (Breiman *et al.*, 1984) that the resubstitution estimate of classification error (*ie:* based on the training set) decreased to zero as the tree size was allowed to increase, until eventually each terminal node contained only a single training sample. As this increase in size occurred, however, the error on an independent test set increased. The method used by CART to arrive at a good-sized tree is to allow an unreasonably-large tree to grow, then prune it back to improve generalisation.

Splits are found by exhaustive search in CART:

```
1. for each variable $x_i$
    (a) if $x_i$ is an ordered variable
          • if $x_i$ only takes on $N$ distinct values in the training set, $x_i \in \{x_{i1}, x_{i2}, \dots x_{iN}\}$,
            then for each value $x_{ij}$ generate the split:
                $x_i \leq x_{ij} + \frac{x_{i(j+1)} - x_{ij}}{2}$
        else $x_i$ is an enumerated variable
          • for each sub-set $B$ of the set of values $A$ that can be taken by $x_i$, generate
            the split:
                $x_i \in B$
        end if
    (b) Select the best split for this variable.
   end for
2. Select the variable with the best split.
```

The concept of which split S is best at node t is determined by the goodness-of-split criterion $\phi(S,t)$. There are many different criteria, but all are based on the concept of *purity*: the more homogeneous the samples are at a node in terms of classes, the more pure the node. Therefore if a node t has impurity $i(t)$, a split that results in the two child nodes t_L and t_R results in a decrease in impurity:

$$\delta i(S,t) = i(t) - p_L.i(t_L) - p_R.i(t_R)$$

where p_L and p_R are the proportions of the sub-set at t going to the left and right child nodes. The split with the highest decrease in impurity is best.

Examples of impurity measures are:

entropy:

$$i(t) = -\sum_{j=1}^{C} p(j|t)\log[p(j|t)]$$

Gini index:

$$i(t) = 1 - \sum_{j=1}^{C} p^2(j|t)$$

A simple method for halting of the splitting process is to set a threshold β for $\delta i(S,t)$, and stop when $\delta i(S,t) < \beta$. Experimentation has shown that this method is not very robust to the selection of β (Breiman *et al.*, 1984). When pruning is used, the size of the initially-grown tree is not such a concern. In fact, if sufficient computational resources were available, the best option would be to continue until the largest possible tree size is reached. The algorithm used in practice is to stop splitting a node t when:

- t is pure; *ie:* each sample in the node belongs to the same class; or
- the number of samples $N(t) \leq N_{min}$, a user-defined minimum-number of samples required for a split; or
- all the data samples in t are identical.

Define a *pruned sub-tree* T' of the tree T to be the tree remaining when the branch T_t rooted at some node t is removed; the notation is $T' \prec T$. The branch is pruned by removing the sub-tree rooted at t but leaving the node t in place. We write:

$$T' = T - T_t$$

The pruning method proceeds by iteratively pruning the weakest link in the tree. This results in a sequence of sub-trees of the original large tree T_{max}, all of different sizes. The best-sized sub-tree is then selected using an "honest estimate" of the misclassification rate.

The error rate used to prune the tree is based on the training set error (resubstitution error). Since this would always be biased towards larger trees, it is used in the context of *minimal cost-complexity* pruning. Defining the complexity of $T \preceq T_{max}$ to be the number of terminals $|\tilde{T}|$ in T, the cost-complexity measure is:

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}|$$

where $R(T)$ is the resubstitution error of tree T. For each $\alpha \geq 0$, we seek the minimum cost-complexity sub-tree $T(\alpha) \preceq T_{max}$ which satisfies:

$$\begin{array}{l} R_\alpha(T(\alpha)) = \overset{min}{T \preceq T_{max}} R_\alpha(T) \\ \text{if } R_\alpha(T) = R_\alpha(T(\alpha)), \text{ then } T(\alpha) \preceq T \end{array}$$

For any node $t \in T$ which is the root of a sub-tree T_t, we have:

$$\begin{aligned} R_\alpha(t) &= R(t) + \alpha & (2.14) \\ R_\alpha(T_t) &= R(T_t) + \alpha|\tilde{T_t}| & (2.15) \end{aligned}$$

For sub-tree T_t to be of use, we need $R_\alpha(T_t) < R_\alpha(t)$. There will be some critical value of α for which equality is obtained, and the node t becomes preferable to the sub-tree below it. Solving Equations(2.14) and (2.15):

$$0 < \alpha < \frac{R_\alpha(t) - R_\alpha(T_t)}{|\tilde{T_t}| - 1} \qquad (2.16)$$

The right-hand side of Inequality 2.16 can be calculated for each node in T, marking equality with α. The sub-tree with the lowest value of this threshold α is the weakest link, in the sense that as α increases, it is the first sub-tree for which its root node becomes preferable, and should be pruned first.

The pruning algorithm starts with T_{max}. First, every node t' for which $R(t') = R(t'_L) + R(t'_R)$ is pruned, resulting in the tree $T_1 = T(\alpha = 0)$. Then, calculate the function:

$$g_1(t) = \begin{cases} \frac{R_\alpha(t) - R_\alpha(T_t)}{|\tilde{T}_t| - 1} & t \notin \tilde{T}_1 \\ +\infty & t \in \tilde{T}_1 \end{cases}$$

Prune all sub-trees $\bar{t}_1$ for which:

$$g_1(\bar{t}_1) = \overset{min}{t \in T_1}\; g_1(t)$$

to obtain T_2. Repeat this process starting with T_2, and so on, to arrive at a sequence of progressively-smaller trees, eventually ending with the root node only:

$$T_1 \succ T_2 \succ T_3 \succ \ldots \succ t_1 \tag{2.17}$$

In practice, this algorithm tends to prune many nodes in the beginning, and progressively fewer towards the end.

There are several interesting and convenient properties of the quantities in this process. If we let $\alpha_k = g_k(\bar{t}_k)$, then it can be shown that $\alpha_k < \alpha_{k+1}\ \forall\ k \geq 1$ in the sequence of trees. Also, due to the discrete nature of the experiment, the tree $T(\alpha_k)$ remains the minimum cost-complexity tree for the range of continuous values between α_k and α_{k+1}.

Given the sequence of trees in Equation 2.17, it remains to select the best pruned sub-tree based on some independent measure of classification error. The two most widely used methods are an independent validation sample, or v-fold cross-validation. The first method is straight-forward, the second more complex. For cross-validation, we select a number V and divide the training set L into V disjoint sub-sets L_v. The vth learning sample is $L^{(v)} = L - L_v$. Then V auxiliary trees are grown on each respective L_v along with the main tree on L. For each value of α we have $T(\alpha)$ and $T^{(v)}(\alpha), v = 1, \ldots, V$, the minimum cost-complexity sub-trees of T_{max} and $T^{(v)}_{max}$. For each cut-off value α_k of T_k, the geometric mean of the continuous range of α is used as the frozen value for combining the auxiliary trees:

$$\alpha'_k = \sqrt{\alpha_k.\alpha_{k+1}}$$

Then the errors of the $T^{(v)}(\alpha'_k)$ on the respective L_v are combined to form an estimate of the true error rate $R^{cv}(T_k)$. The sub-tree $T(\alpha_{k0})$ derived from L that minimises this error estimate is chosen as the best-sized sub-tree:

$$R^{cv}(T_{k0}) = \overset{min}{k}\; R^{cv}(T_k) \tag{2.18}$$

The analysis of the bias of this method is complicated, but errs on the conservative side of larger errors since the $T^{(v)}(\alpha)$'s are derived from a smaller training sample and are therefore generally less accurate.

The rule of Equation 2.18 can be modified to take into account the variability in results with cross-validation error due to different random division into the V sub-sets. The new rule, called the *1 S.E. Rule*, assumes a binomial distribution for the errors of the classifier, and selects the smallest tree with misclassification rate estimate within one standard error of the lowest:

$$\text{Select tree } T_{k1} \text{ with } k1 \text{ the maximum } k \text{ such that } \hat{R}(T_{k1}) \leq \hat{R}(T_{k0}) + \text{SE}(\hat{R}(T_{k0}))$$

As was stated earlier, the absence of a variable in the splits does not indicate that it has no correlation with the class of the samples. A method of ranking the input variables in order of importance is given in (Breiman *et al.*, 1984). The method is based on the surrogate splits $\tilde{S}_m$ of variable x_m. Let T be the optimal tree obtained through growing and pruning. Compute the loss of impurity of the surrogate split, $\delta I(\tilde{S}_m, t)$, at each node in T. The importance measure is defined as:

$$M(x_m) = \sum_{t \in T} \delta I(\tilde{S}_m, t)$$

Therefore similar splits of which only one can be chosen for the final tree will offer similar contributions to their respective importance values. The importance measure is usually normalised to $100.M(x_m)/ \overset{max}{m} M(x_m)$ so that the most important variable has a value of 100, and the others range from 0 to 100.

2.8.2 C4.5

C4.5 is an extension of the ID3 algorithm; both were developed by Ross Quinlan (Quinlan, 1993). The two main characteristics of the C4.5 algorithm are that a single attribute can only be used once in any given path from the root to a leaf node, and the splitting criterion used is information gain.

For a discrete attribute A with n possible values, the splitting rule results in n outcomes (tree branches). For a real-valued variable, there are only two outcomes of the test $A \leq t$. Each pair of adjacent values for the variable in the test set is a possibility for a split point t. At each step, that variable is chosen which maximises the *gain ratio*:

$$\text{Gain Ratio}(D,T) = \frac{\text{Gain}(D,T)}{\text{Split}(D,T)}$$

Here D is the set of samples in question, T is the proposed test with k outcomes, C is the number of classes and $p(D,j)$ is the proportion of cases in D that belong to class j. From information theory, the uncertainty about the class to which a case in D belongs is (Quinlan, 1996):

$$I(D) = -\sum_{j=1}^{C} p(D,j) \times \log_2(p(D,j))$$

The information gained by test T is:

$$\text{Gain}(D,T) = I(D) - \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times I(D_i)$$

This quantity tends to increase with the number of outcomes and therefore favours discrete variables that have many values, so the gain ratio is formed by using the split information:

$$\text{Split}(D,T) = -\sum_{i=1} k \frac{|D_i|}{|D|} \times \log_2 \left(\frac{|D_i|}{|D|} \right)$$

which is the potential information obtained by simply dividing the data into k sub-sets, regardless of class.

Continuous attributes tend to be favoured when selecting splits: for an attribute with N values in the training set, there are $N-1$ splits to choose from, whereas for a discrete variable there is only one possible split. Two modifications to C4.5 are described in (Quinlan, 1996) to overcome this problem:

1. the *minimum description length* principle is the minimisation of the total *message length*, which is the sum of the bits required to encode the decision tree (the theory)

and the bits needed to correct the errors made by the tree (the exceptions). The MDL principle provides a mechanism for trading off the complexity and accuracy of the tree to preserve generalisation. A split on a discrete variable requires the encoding of the variable identifier only, whereas for a continuous variable an additional $\log_2(N-1)$ bits are needed to encode the split point t. This increase in message length is subtracted from the information gain on a per-case basis:

$$\text{Gain}'(D,T) = \text{Gain}(D,T) - \log_2(N_T - 1)$$

Now a continuous attribute with many observed distinct values is less likely to be selected.

2. the split information varies with the split position t and is maximal when there are as many cases in D above t as there are below. Since the number of outcomes is always two for a continuous variable, the gain rather than the gain ratio is used to select the split point. Note that the gain ratio is still used to select the best variable.

Noisy data often leads to over-sized trees, so pruning is used.

2.8.3 QUEST

The QUEST algorithm (*Quick, Unbiased, Efficient Statistical Trees*) is a decision tree algorithm based on the FACT and CART methods which addresses the following issues (Loh and Shih, 1997*b*):

- exhaustive search methods tend to be biased toward variables with more possible splits; and
- the order of computational complexity for a categorical variable with N values is exponential, $\mathcal{O}(2^{N-1} - 1)$.

The FACT algorithm deals with the problems of variable selection and split selection separately. To select the split variable, the analysis-of-variance F-statistic is calculated for continuous variables, and the variable with the largest statistic is selected. Linear discriminant analysis is then used to find the split point. Discrete variables are handled by transforming them into ordered variables using the CRIMCOORD method. The FACT algorithm results in one sub-tree for each class represented at the node.

In contrast, Quest yields binary splits by applying a two-means clustering algorithm and then using quadratic discriminant analysis to determine the split point between the two super-classes. The FACT method of variable selection is biased towards discrete variables, because after the CRIMCOORD transformation they tend to be stochastically larger. The QUEST method overcomes this bias by using a different statistical test for discrete variables; the overall process is quite complicated.

The QUEST algorithm can be used with v-fold cross-validation, pruning and linear combination splits. A recent comparison of several decision tree methods on public databases has found that QUEST performs relatively well overall in terms of accuracy, speed and tree size (Lim *et al.*, 1997).

2.9 Feature Selection

Feature selection is the process of selecting a sub-set $\mathcal{X}'$ of the original set of input measurements (features) $\mathcal{X}$ to use for classification. One might argue that the maximum number of features in hand should be presented to the classifier, since this provides the maximum amount of information available about the problem. It has been observed, however, that there is a "peaking" effect associated with the addition of features to a classifier: as features are added, performance improves up to a point, after which additional features degrade performance (Young and Fu, 1986). Also, the features are usually correlated, so in situations where an economy of sensors or classifier size is required, redundant features should be removed.

Although there could be several reasons for the peaking effect previously mentioned, the main reason is the *curse of dimensionality* (Duda and Hart, 1973), the principle that as the dimensionality of data points increases, they become more sparse in the feature space. In fact, the hyper-volume of the input space increases exponentially as more features are added (Sinkkonen, 1998). Since a classifier must provide a covering over the input space, and the amount of resources required are in some way proportional to the hyper-volume of the space, classifier generalisation performance tends to degrade as dimensionality is increased.

For a general classifier, the optimal features cannot be selected independently of one another since they may be correlated. The only way to pick the optimal sub-set of features is to perform an exhaustive search over all possible sub-sets and optimise some criterion. For a set of d features, there are $2^d - 1 \approx 10^{0.3\times d}$ possible sub-sets, prohibiting exhaustive search for a significant number of features. For instance, if the problem is prediction for financial forecasting and we wish to find the best set of stock indicators for our problem, there may be thousands of possible indicators, from the Dow Jones to the humidity in Namibia. In cases where there are many features to choose from, a heuristic search for a good sub-set must be performed.

The next question is what criterion to use during the search. The obvious choice is to examine the performance of the classifier directly using the sub-set of features at hand. This approach is usually too computationally expensive, and the solution obtained may be more characteristic of the particular training and test sets rather than the whole population (Ripley, 1996). If the class-conditional density functions are known, then the true error probabilities can be used to select the best sub-set. For example, the *divergence* criterion for two classes is (Young and Fu, 1986):

$$J_D(\mathcal{X}') = \int_{\mathbf{x}\in\mathcal{X}'} [p(\mathbf{x}|c=1) - p(\mathbf{x}|c=2)].\ln\left[\frac{p(\mathbf{x}|c=1)}{p(\mathbf{x}|c=2)}\right].d\mathbf{x}$$

J_D is zero when the class distributions fully overlap, and a maximum when there is no overlap at all. For the multi-class case, the criterion is averaged over all class-pair combinations:

$$J_D(\mathcal{X}') = \sum_{i=1}^{C} \sum_{j=i+1}^{C} P(i)P(j)J_{ij}(\mathcal{X}')$$

Shannon's entropy measure can also be used:

$$J_S(\mathcal{X}') = -\int_{\mathbf{x}\in\mathcal{X}'} \sum_{i=1}^{C} P(i|\mathbf{x}) \log_2[P(i|\mathbf{x})].p(\mathbf{x}).d\mathbf{x}$$

Such criteria simplify when the distributions are assumed to be Gaussian.

Distribution-free criteria that do not involve error probabilities are based on the relationship of within-class distances and between-class distances. These criteria are also used for clustering, or unsupervised learning. Given some distance metric $d(\mathbf{x}_{ik}, \mathbf{x}_{jl})$ between the kth pattern of the ith class and the lth pattern of the jth class, the inter-class distance criterion is:

$$J_d(\mathcal{X}') = \frac{1}{2}\sum_{i=1}^{C} P(i) \sum_{j=1}^{C} P(j) \frac{1}{n_i n_j} \sum_{k=1}^{n_i}\sum_{l=1}^{n_j} d(\mathbf{x}_{ik}, \mathbf{x}_{jl})$$

There are several heuristic search algorithms that have been used for some time to select sub-optimal feature sub-sets. Step-wise strategies assume that the feature selection criterion is monotonic:

$$\text{if } \mathcal{X}_j \text{ is a sub-set containing } j \text{ features, } \mathcal{X}_1 \subset \mathcal{X}_2 \subset \ldots \mathcal{X}_d,$$

$$\text{then } J(\mathcal{X}_1) \leq J(\mathcal{X}_2) \ldots \leq J(\mathcal{X}_d)$$

Forward selection is the process of starting with one feature, then at each step adding the feature that maximises the increase in J (Ripley, 1996). *Backward selection* is the opposite process: we start with the full set of features and iteratively remove the feature whose presence least increases J. These two methods are rather limited in that they do not account for interactions between features.

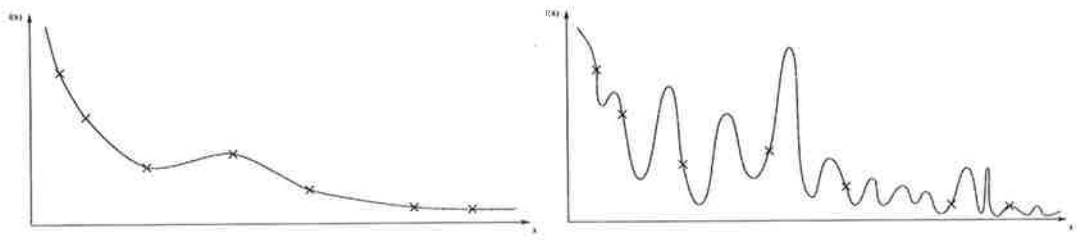
A popular method from combinatorial optimisation that does produce optimal results with respect to the criterion used is the *branch and bound* algorithm. Savings are made during the search using the monotonicity property by eliminating sub-set A from further investigation if we know that a larger sub-set $A' \subset A$ has a value of J that is below our current best value α over sub-sets of size k. We start with an estimate of α obtained using one of the other heuristic techniques, and consider the set of all features. We recursively drop one feature at a time to form branches in a search tree. At any point, the sub-tree below a sub-set with a J value lower than the best so far need not be considered further. A sub-set with J higher than the best is used to update α.

Genetic algorithms have also been applied to the task of feature selection (Punch *et al.*, 1993; Vafaie and Imam, 1994; Vafaie and De Jong, 1993). Several criteria have been developed to measure the relevance and irrelevance of features for classification (John *et al.*, 1994). A review of feature selection methods is given in (Scott *et al.*, 1998), where it is concluded that in order for a feature selection algorithm to be successful, the classifier to be used with the features must be taken into consideration.

2.10 Generalisation and Model Complexity

For a given set of data there are an infinite number of possible explanations or models, and those models will vary in their complexity. For instance, if we were both to see a dolphin balance a ball on its snout, you may conclude that it has been trained by humans to do so, whereas I might postulate in a somewhat contrived manner that the dolphin is actually a visitor from a parallel universe with an intellect far superior to ours, and is trying to communicate to us its desire for a tin of dubbin. The general principle that the simplest explanation is true offers widespread appeal to human beings. In science, this principle is known as *Occam's Razor*, and was popularised by William of Ockham(1285-1349). What William actually said was "Pluralitas non est ponenda sine necessitas" (plurality shouldn't be posited without necessity) (Ellison, 1995). He was by no means the first to express the concept, since similar statements are found in the writings of his teacher Duns Scotus. Even Aristotle in the Physics (book I, chapter vi) wrote "for the more limited, if adequate, is always preferable".

This issue is pertinent for classification, since the objective is to construct a classifier that generalises well. The classifier interpolates the given set of training points, and the behaviour of the interpolation between these points determines the quality of the generalisation. There is an infinite number of ways in which to do this, but if the training data is on the same scale as the structure of the underlying model, then the smoothest or most regular mapping is desirable. Consider, for example, the interpolation problem of Figure 2.13. Both interpolations are correct in that they pass through the points provided, but the second is condemned in the observer's mind as inplausible because it is not a well-behaved interpolator.



(a) "smooth" interpolation

(b) "non-smooth" interpolation

Figure 2.13: An example of two possible interpolations of a set of measurements.

The consequence for classification is that classifiers which are less complex will tend to generalise better. Aside from splitting ties between equally-performing classifiers, we may want to cut out the cross-validation process altogether and construct a classifier from a training set alone using the principle of parsimony. This would require the minimisation of some function of model complexity and model error. Difficulties soon arise over how to quantify the complexity of the model and in what proportions to trade-off the two quantities. There is even no reason to believe that a linear combination should be used.

The Bayesian framework allows this trade-off to occur in a principled manner, since a more complex model is more likely to have a lower conditional probability (Hanson *et al.*, 1991). Another mechanism that is more explicit about model complexity is the *minimum message length* (MML) principle (Quinlan, 1989), also known as *minimum description length* (MDL). MML and MDL were developed independently but are virtually identical. The MML principle is usually stated as the following communication problem: I have a set of data D which I intend to communicate to you as a string of bits in the most concise way possible. We agree prior to this communication on an encoding scheme for the data; this scheme remains fixed and is independent of the data. Now I could transmit the data to you directly, but it would probably contain redundant information. Instead, I develop a model for the data, M, and transmit the model and the data given the model. The total length of this message is $L(M) + L(D|M)$. The principle states that the model that minimises the bit-length of this message is the best model. For example, the model portion of the message may be the parameters for the distribution of the data, while the data may be transmitted as a Huffman code that was constructed based on this distribution. Although the problem is framed in this way, the most important aspect is the *model* rather than the bit-string communicated: therefore it is only necessary to be able to calculate the *encoded length* of the model and message, rather than the actual encoded strings themselves. The MML principle has obvious applications in data mining and physics, since it generates a likely explanation for a set of data.

For classification, consider the use of decision trees as models. You and I both have a copy of the data set, but only I have the class labels which I wish to transmit to you. I construct a decision tree to classify this data, but due to class overlap there will be some errors. Therefore the message I send to you is the encoded tree plus the exceptional cases which the tree classifies incorrectly (Quinlan, 1989). The tree that minimises the length of this message is the MML-optimal tree. One of the main issues in the use of MML is how to efficiently encode the model and the data; in general the encoding is sub-optimal which results in a sub-optimal tree.

It has long been the goal of machine learning researchers to devise the perfect classifier which can out-perform all other methods on all problems. Therefore one finds many general classifiers in the literature that are designed to operate on any data regardless of its origin. Researchers have, however, reached the gradual realisation that there is no "Holy Grail" of classifiers, but rather some methods are more appropriate than others in particular situations. An important result is presented in (Schaffer, 1994): a conservation law for generalisation performance.

Theorem 2.2 (Law of Conservation of Generalisation Performance) *Define a learning situation S as a triple (D, C, n) where D and C specify how the data are generated and n is the size of the training set. Define $GP_L(S)$ to be the generalisation performance of learner L in learning situation S. For any learner L,*

$$\sum_S GP_L(S) = 0$$

The theorem was proven for two-class problems. Generalisation performance was defined as the correct classification rate of the learner on examples not used in the training set minus 0.5. The conservation law has interesting consequences for "general" classifiers. For instance, it is possible for a classifier to perform very well on a few learning situations and mildly worse-than-chance on the rest of learning situations, or to perform very well on a sub-set of situations, very badly on another sub-set and neutrally on the rest of the possible learning situations. It is not, however, possible for a classifier to perform better than chance in all learning situations, or to have better-than-guessing performance on a sub-set of situations and neutral performance on the rest. Good performance in some situations must necessarily be balanced by poor performance in others. A similar result has been obtained in the *No Free Lunch* theorems for learning algorithms (Wolpert, 1996), which conclude that any two learning algorithms have the same expected generalisation loss when averaged over all target functions.

The conservation of generalisation shifts the focus of research from "the quest for the Holy Grail", to the question of which algorithms perform best in certain situations. Another consequence is that any demonstration that an algorithm A is better than another algorithm B on some test suite is counter-balanced by inverted relative performance on some other test-suite.

2.11 Benchmark Standards

There is no question that pattern recognition is an exciting field that captures imaginations with its many possibilities and futuristic applications. Perhaps this coupled with the ease with which classifiers can be implemented on a computer has sparked the proliferation of publications about new methods or applications for classification. Just as computer hardware developments have succeeded advances in computer software, the theoretical analysis of learning algorithms and the principled and methodical presentation of their results has fallen far behind the number of publications involving superficial empirical investigations. A study of 190 neural network articles published in 1993 and 1994 revealed that 29% contained no realistic learning problems, only 8% presented results for more than one real-data problem, and one third of papers didn't contain a comparison with other known algorithms (Prechelt, 1996). Another study (Flexer, 1996) on 61 articles from neural network journals found that only 43% of experiments involved multiple training runs, only 5% used a statistical test in a comparison with another method, and although 72% used a hold-out test set, only 5% used a validation set.

In addition to these findings, it is rare to find the full details of algorithm parameters and experimental configuration in a publication, and those parameters that are specified are sometimes ambiguous in their meaning. Furthermore, reliance of results on the permutation, partitioning and encoding of the data often makes a direct comparison with published results impossible.

There have been several attempts to overcome these problems by standardising experiments and comparisons (Prechelt, 1994; Flexer, 1996; Zheng, 1993; Rasmussen *et al.*, 1996; ESPRIT, 1995). Many of the conventions adopted in this thesis have been taken from the Proben1 benchmark report (Prechelt, 1994), and will be described in Chapter 6. Of particular interest in benchmarking are statistical tests for the comparison of two or more classification algorithms. Some recommended methods are described next.

2.11.1 McNemar's Test

Suppose we have two classifiers and we wish to compare their performance on the test set. Simply assuming the errors are binomially distributed and comparing the error bounds on the two error rates gives a pessimistic test of their difference, since there are some samples that both algorithms always get wrong or right, inflating both variances. McNemar's Test (Ripley, 1994), a statistical test for differences between proportions in paired sample designs, overcomes this problem. Let the two algorithms be A and B, with misclassification probability estimates e_A and e_B on some test set. We construct the null hypothesis that these two algorithms are equivalent:

$$H_0 : e_A = e_B$$

Without loss of generality, we want to test H_0 against:

$$H_1 : e_A < e_B$$

where A is the algorithm that resulted in fewer misclassifications. We then construct the contingency table for the number of samples correctly and incorrectly classified by both algorithms, shown in Table 2.3. The main-diagonal elements contain the number of samples that both algorithms agreed on. The important elements come from the off-diagonal: these describe the differences between the algorithms.

Table 2.3: Example contingency table for methods A and B.

	C_B	I_B	Total
C_A	(C_A, C_B)	(C_A, I_B)	(C_A)
I_A	(I_A, C_B)	(I_A, I_B)	(I_A)
Total	(C_B)	(I_B)	m

Let (I_A, C_B), the number of observations correctly classified by B but not by A, be defined as success, and let $M = (I_A, C_B) + (C_A, I_B)$. Under the null hypothesis, the number of successes is binomially distributed:

$$(I_A, C_B) \sim B(M, 0.5)$$

For a given problem, we can use the cumulative probability density function for $B(M, 0.5)$ at (I_A, C_B) to calculate the level of significance of the result:

$$\alpha = F(x \leq (I_A, C_B))$$

The probability that H_0 can be rejected is $(1 - \alpha)$.

Note that McNemar's test gives a measure of the statistical significance that algorithm A is better than B, but doesn't say by how much.

2.11.2 t-Test

The t-Test can be used to test for a significant difference in the *average* error rates of two classifiers over several runs. Given the mean $\bar{x}$ and sample standard deviation s of an error rate estimate for a classifier obtained from a sample of N values, we can estimate the standard error of the mean:

$$\hat{\sigma}_{\bar{x}} = \frac{s}{\sqrt{N}}$$

For sufficiently large values of N, $\bar{x}$ is distributed normally with mean $\bar{X}$ and standard deviation $\hat{\sigma}_{\bar{x}}$, so that the true mean error rate $\bar{X}$ lies in the confidence interval $\bar{x} \pm 1.96\hat{\sigma}_{\bar{x}}$ with probability 0.95.

For small samples (*ie:* $N < 30$) the normality assumption is no longer valid. Instead, a Student's t-distribution must be applied which yields accordingly larger confidence intervals. A statistically-significant result can only be guaranteed if the two confidence intervals do not overlap. Otherwise, a t-test for the significance of the difference between means must be used. We have two classification algorithms, A and B and we perform N_A and N_B runs on A and B respectively. It is assumed that the error rates both come from normal distributions with the same standard deviation but different means. Using the sample means $\bar{x}_A$ and $\bar{x}_B$ and variances s_A^2 and s_B^2 of the runs, we perform the following test.

$$\begin{aligned}
t_{\bar{x}_A - \bar{x}_B} &= \frac{\bar{x}_A - \bar{x}_B}{\hat{\sigma}_{\bar{x}_A - \bar{x}_B}} \\
\hat{\sigma}_{\bar{x}_A - \bar{x}_B} &= \sqrt{\hat{\sigma}^2_{pooled}\left(\frac{1}{N_A} + \frac{1}{N_B}\right)} \\
\hat{\sigma}^2_{pooled} &= \frac{(N_A - 1)s_A^2 + (N_B - 1)s_B^2}{N_A + N_B - 2}
\end{aligned}$$

With the t-statistic, the hypothesis that A and B give the same error rate can be rejected at some level of significance, say $\alpha = 0.05$ by looking in a t-table (for a two-tailed test) with $(N_A + N_B - 2)$ degrees of freedom.

In general, the populations of errors generated by the two classifiers will have different variances. The statistic for the unequal variance test is (Press *et al.*, 1992):

$$t = \frac{\bar{x}_A - \bar{x}_B}{\sqrt{s_A^2/N_A + s_B^2/N_B}}$$

t is approximately distributed as a Student's t distribution with a number of degrees of freedom equal to:

$$\nu = \frac{\left(\frac{s_A^2}{N_A} + \frac{s_B^2}{N_B}\right)^2}{\frac{(s_A^2/N_A)^2}{N_A - 1} + \frac{(s_B^2/N_B)^2}{N_B - 1}}$$

2.11.3 Comparing Multiple Classifiers

The simplest way of comparing multiple classifiers is to perform each distinct pairwise comparison. Given the confidence α, the probability of making at least one Type I error in a family of J statistically-independent tests is the family-wise error rate, $FWE = 1-(1-\alpha)^J$; for $\alpha = 0.05$, $FWE = 0.64$. One testing method that overcomes this problem is called the *one-way repeated measures design*, and is described in (Feelders and Verkooijen, 1995). It produces confidence intervals for the pairwise differences in error rates between the methods, and compensates for the multiplicity effect. Given the n_{tst} test set observations and the k classification methods $f_1, \ldots, f_k$, the first step is to produce the one-way repeated measures lay-out, shown in Table 2.4. Y_{ij} is 1 if f_j classifies observation i correctly, and zero otherwise.

Table 2.4: One-way repeated measures lay-out.

	Functions						
Observations	f_1	f_2	$\dots$	f_j	$\dots$	f_k	Total
1	Y_{11}	Y_{12}	$\dots$	Y_{1j}	$\dots$	Y_{1k}	$Y_{1.}$
2	Y_{21}	Y_{22}	$\dots$	Y_{2j}	$\dots$	Y_{2k}	$Y_{2.}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
i	Y_{i1}	Y_{i2}	$\dots$	Y_{ij}	$\dots$	Y_{ik}	$Y_{i.}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	Y_{n1}	Y_{n2}	$\dots$	Y_{nj}	$\dots$	Y_{nk}	$Y_{n.}$
Total	$Y_{.1}$	$Y_{.2}$	$\dots$	$Y_{.j}$	$\dots$	$Y_{.k}$	
Means	$\bar{Y}_{.1}$	$\bar{Y}_{.2}$	$\dots$	$\bar{Y}_{.j}$	$\dots$	$\bar{Y}_{.k}$	

The pooled variance for each pairwise difference $\bar{Y}_{.j} - \bar{Y}_{.j'}$ is:

$$\hat{\sigma}^2_{\text{diff}} = \frac{2(k\sum_{i=1}^{n_{tst}} Y_{i.} - \sum_{i=1}^{n_{tst}} Y_{i.}^2)}{n_{tst}^2 k(k-1)}$$

Using the notation that θ_j is the proportion of correct classifications on the test set of classifier f_j, the $100(1-\alpha)\%$ simultaneous confidence intervals for all pairwise differences $\theta_j - \theta_{j'}$ is:

$$\theta_j - \theta_{j'} \in \left[\bar{Y}_{.j} - \bar{Y}_{.j'} \pm Z^{\nu}_{k^*:1-\alpha/2}.\hat{\sigma}_{\text{diff}}\right] \quad (1 \leq j < j' \leq k)$$

where $Z^{\nu}_{k^*:1-\alpha/2}$ is based on the Student t distribution adjusted for k^*, the total number of pairwise comparisons involved, and $\nu = n_{tst} - 1$ the degrees of freedom. Tables for the Z statistic can be found in (Marascuilo and McSweeney, 1977). The value for Z increases with k^*, leading to wider confidence intervals. For each data set, the method results in a table of confidence intervals for each pair of methods. Intervals that contain 0 indicate that there is no significant difference between the errors of the two classifiers.

2.12 Conclusion

This chapter has presented an introduction to pattern recognition, and a comprehensive overview of supervised classification. Material presented later in this thesis refers back to this chapter. The next chapter has a similar function, but for genetic programming instead of pattern recognition. With the introductory material aside, the main points of this thesis begin in Chapter 4.

Chapter 3

Evolutionary Optimisation Techniques

3.1 Introduction

The previous chapter has presented a founding knowledge of pattern recognition, and in particular those topics that are relevant to the work of this thesis. This chapter serves the same purpose for the topic of evolutionary computation. First, a general overview of optimisation is given. Then evolutionary computation is introduced as a set of algorithms inspired by the principles of biological evolution. Finally those paradigms of particular relevance to this thesis, genetic algorithms and genetic programming, receive a lengthier treatment. The reader familiar with these areas may wish to skip this chapter.

3.2 Optimisation and Search

Many problems of interest to humans can be framed as optimisation problems. An optimisation problem requires an *objective function* which quantifies how good or bad a solution is. Such a function maps the space of possible solutions $x \in \mathcal{X}$ into a real scalar value, $f(x) : \mathcal{X} \to \Re$. The term "search" is synonymous with optimisation because optimisation is really a search for a global minimum or maximum of the objective function. An *optimisation algorithm* is a series of steps which are carried out to search for an extremum of the objective function. Two examples of an optimisation problem are:

1. Find the minimum value of (Foulds, 1981):

$$f(x, y) = 4x + 3y$$

   subject to the constraints:

$$\begin{aligned} 3x + 4y &\leq 12 \\ 4x + 2y &\leq 8 \\ 0 &\leq x \text{ and} \\ 0 &\leq y \end{aligned}$$

2. The *Traveling Salesman Problem* (Telfar, 1994): given the $n \times n$ matrix $C = [c_{ij}]$ where c_{ij} is non-negative and denotes the Euclidean distance separating cities i and j, find the cyclic permutation π of the integers $1, \ldots, n$ that minimises $f(\pi) = \sum_{i=1}^{n} c_{i\pi(i)}$.

There are many differences between these two problems, the most notable being that the first problem is a *continuous* optimisation problem because it has a continuous solution space, while the second is a *combinatorial* optimisation problem because the solution space is discrete.

The objective function is commonly envisaged as a generalised landscape, containing mountains, valleys and other structures in a high-dimensional space. An example of a 2-dimensional objective function is shown in Figure 3.1. For a maximisation problem, the "highest peak" in the landscape is the *global optimum*, the exact solution to the problem. For a minimisation problem, the global optimum is the lowest point in the landscape.

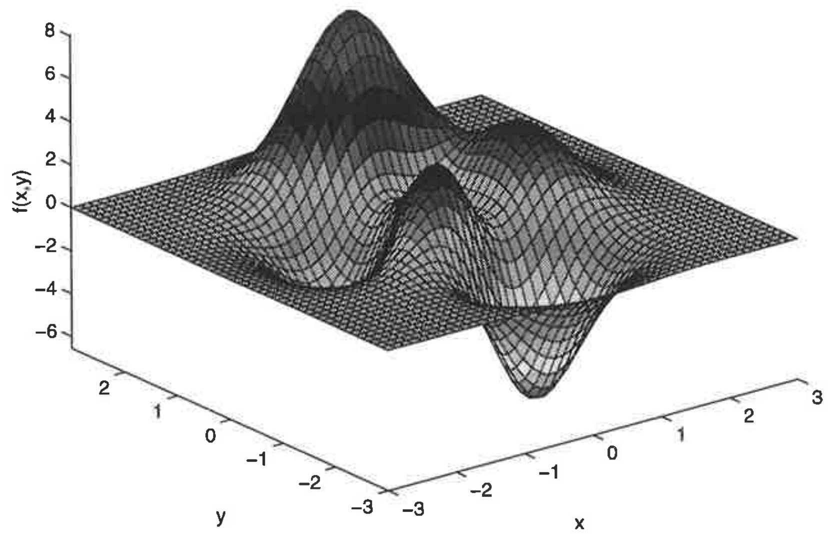


Figure 3.1: An example of a 2-dimensional function $f(x, y)$ viewed as a landscape.

3.2.1 Complexity Theory

Another difference between the example problems of the previous section is that the first is considerably easier to solve than the second. *Complexity theory* is concerned with the amount of computation required to solve a problem (Goldschlager and Lister, 1988). Complexity theory uses *order notation* to bound the computational complexity of an algorithm. An algorithm is said to be *order p(n)* ($\mathcal{O}(p(n))$), where n is some measure of the size of the problem and $p(.)$ is some function, if the number of steps required by the algorithm to solve the problem is bounded by a constant times $p(n)$. Problems are divided into two categories: those that are "easy", and those that are "hard" (Weisstein, 1998). Easy problems can be solved exactly in polynomial-time, and are said to be in the set $\mathcal{P}$. A polynomial-time algorithm has a number of steps that is bounded by a polynomial function of the size of the problem. Linear programming, a numerical technique used to solve the first problem above, is known to be in $\mathcal{P}$.

Some problems are so hard as to be *undecidable*, and cannot be solved by *any* algorithm. Examples of undecidable problems are the halting problem, Hilbert's tenth problem (solvability of polynomial equations in integers) and several problems of tiling the plane (Garey and Johnson, 1979). The remaining hard problems require an exponential-time algorithm for their solution, and are said to be *intractable* or *infeasible*.

Problems for which no polynomial-time algorithm exists fall into two classes: those that are proven to be hard, and those that are thought to be hard but have not been proven to be so. Of those unproven hard problems, there is a class that can be solved in polynomial time on a non-deterministic computer called the class of $\mathcal{NP}$ (*non-deterministic polynomial*) problems (Garey and Johnson, 1979). If an algorithm $A \in \mathcal{NP}$ and every problem in $\mathcal{NP}$ can be transformed to A via a polynomial-time algorithm, then A is said to be *$\mathcal{NP}$-complete* (Telfar, 1994). The traveling salesman problem is an example of an $\mathcal{NP}$-complete problem. This class of problems includes the "hardest" problems in $\mathcal{NP}$, and is particularly interesting because if one $\mathcal{NP}$-complete problem were proven to be $\mathcal{P}$, then it is a consequence that all $\mathcal{NP}$-complete problems are $\mathcal{P}$. It is generally believed that $\mathcal{NP}$-complete problems are not solvable in polynomial time. There are hundreds of problems known to be $\mathcal{NP}$-complete, such as the Hamiltonian path problem, the Steiner tree problem, the bin packing problem and various scheduling tasks. An extensive list is found in (Garey and Johnson, 1979).

If a problem A is not known to be in $\mathcal{NP}$ but every problem in $\mathcal{NP}$ is polynomial-reducible to it, then A is said to be *$\mathcal{NP}$-hard* (Telfar, 1994), that is at least as hard as any $\mathcal{NP}$ problem (Weisstein, 1998).

3.2.2 Heuristic Search Algorithms

Many interesting optimisation problems are $\mathcal{NP}$-complete or $\mathcal{NP}$-hard. For these problems, an exact solution cannot be obtained in polynomial time: often the only way to obtain the guaranteed global optimum is via exhaustive search. In most cases such a recourse is prohibitively expensive, and one must settle with a near-optimal solution in a reasonable amount of time. Algorithms which accomplish this goal are termed *heuristic search* algorithms. Of these algorithms, some are generally applicable while others are tailored to the problem but are not suitable for other problems. The more specific techniques are *strong* methods because they exploit knowledge about the problem, and the general techniques are *weak* methods because little domain knowledge is used (Angeline, 1994). Strong methods generally reach an acceptable solution faster than weak methods.

The simplest general heuristic search techniques employ *local search*, or *hill-climbing* (Telfar, 1994). Given the objective function $f(x)$ and the search space $x \in \mathcal{X}$, define $N(x, T)$, the *neighbourhood* of point x, to be the set of points reachable by applying the transformation $T(.)$ to x. Local search starts with some initial solution and iteratively improves upon it by searching the neighbourhood set for a superior solution. When the neighbourhood set contains no solutions of superior quality, the search is at an end. The problem with local search is that objective functions are generally non-convex, so there is no guarantee that the local extremum reached is the global optimum. The quality of the final solution depends on the starting point; the chances of reaching the global optimum can be increased by re-starting the algorithm at different initial points.

Note that hill-climbing, and indeed every heuristic search algorithm, makes the implicit assumption that the objective function is to some degree smooth, so that local extrema are surrounded by search points with similar objective values. In any case, functions for which this is not true have little chance of being successfully optimised. More complicated general heuristic search algorithms are designed with the following goals in mind:

- to find an acceptable solution as quickly as possible; and
- to escape local extrema.

In general, these objectives are met by adapting to the observed characteristics of the objective function.

3.2.3 Relationship between Search and Pattern Recognition

Optimisation is inextricably involved with pattern recognition. In the case of supervised learning, one may be searching for an appropriate model, or given a model, for that model's optimal parameters. For example, the weights that minimise the error function of an MLP cannot be obtained analytically, but via iterative training. The training process starts with an initial set of randomly-generated weights corresponding to a single point on the error landscape. Training proceeds by iteratively estimating the direction of steepest descent in weight-space with respect to error and stepping in that direction by a certain amount. This down-hill trajectory can land the weights in a locally-optimal position. Therefore there is no guarantee of a globally-optimal solution. Alternatively, an algorithm may be devised to search for an optimal MLP architecture.

Not only do many aspects of classifier design involve optimisation, but often heuristic algorithms are required. For instance, it has been shown in (Blum and Rivest, 1988) that the construction of an optimal 3-node neural network is $\mathcal{NP}$-complete. In (Murthy, 1996), several results are cited on the $\mathcal{NP}$-completeness and $\mathcal{NP}$-hardness of different decision tree induction tasks.

Another similarity between search and classification involves the conservation of performance of general algorithms over all problems. In Section 2.10 the conservation law for generalisation performance of classifiers was discussed. A similar result for optimisation has been obtained in the *No Free Lunch Theorems* (Wolpert and Macready, 1996; Wolpert and Macready, 1997). The theorems establish that for any two search algorithms, better performance over a set of problems is offset by worse performance on another set of problems.

Theorem 3.1 (No Free Lunch Theorem for Optimisation) *Define the search space $\mathcal{X}$, the cost-value space $\mathcal{Y}$, and an objective function $f : \mathcal{X} \to \mathcal{Y}$. The set of all possible objective functions is $\mathcal{F} = \mathcal{Y}^{\mathcal{X}}$. An algorithm visits a time-ordered set of m distinct points $d_m \equiv \{(d_m^x(1), d_m^y(1)), \ldots, (d_m^x(m), d_m^y(m))\}$, where each point is a search-space point and associated cost value. For any pair of algorithms a_1 and a_2:*

$$\sum_{f \in \mathcal{F}} P(d_m^y | f, m, a_1) = \sum_{f \in \mathcal{F}} P(d_m^y | f, m, a_2)$$

where $d_m^y \equiv \{d_m^y(1), \ldots, d_m^y(m)\}$ denotes the ordered set of cost values.

The theorem applies to deterministic and stochastic algorithms. This probabilistic approach uses $P(f) = P(f(x_1), \ldots, f(x_{|\mathcal{X}|}))$ to denote the probability that each $f \in \mathcal{F}$ is the objective function for the optimisation problem at hand, and can be used to specify a class of optimisation problems. $P(d_m^y | f, m, a)$ is the conditional probability of obtaining a particular sample of objective function values under the given conditions, and can be used to measure the performance of an algorithm.

Wolpert and Macready's work also presents a geometrical interpretation of the suitability of a search algorithm for a given problem. The probability of obtaining some sequence of search points is:

$$P(d_m^y | m, a) = \sum_{f \in \mathcal{F}} P(d_m^y | m, a, f) P(f)$$

which can be written as an inner product:

$$P(d_m^y | m, a) = \mathbf{v}_{d_m^y, a, m} \cdot \mathbf{p}$$

where $\mathbf{v}_{d_m^y, a, m}$ is the vector of length $|\mathcal{F}|$ containing the conditional probabilities $P(d_m^y | m, a, f)$, and $\mathbf{p}$ is the vector of the same length containing the objective function probabilities $P(f)$. The inner product form implies that the performance of an algorithm is proportional to its projection onto the objective function probability vector $\mathbf{p}$. That is, an algorithm must be aligned with a particular problem to perform well. The framework also provides measures for the performance of optimisation algorithms.

3.3 Paradigms for Evolutionary Search

A field of biologically-inspired research contemporary with neural networks is *evolutionary computation*. This area of research is concerned with optimisation and search algorithms that mimic the principles of biological evolution. In this section, the basic principles of biological evolution are presented, and each of the four main paradigms within evolutionary computation are introduced. Deeper treatments of genetic algorithms and genetic programming are given later in Section 3.4 and Section 3.5.

3.3.1 Biological Evolution

No scientific theory has so revolutionised both scientific and philosophical thought like the Darwinian theories of biological evolution. Beginning with the work of Charles Darwin (1809-1882) and Alfred Russel Wallace (1823-1913) in 1858, the classical evolutionary theory has been refined to the universally accepted paradigm of neo-Darwinism we have today (Fogel, 1995). This paradigm asserts that the enormous variation in life on the earth is accounted for by statistical processes acting on populations of organisms and their genetic codes. The four processes are reproduction, mutation, competition and selection. Organisms compete with one another for limited resources. Those organisms which are more adapted to their environment and are more able to reproduce pass on their beneficial genetic material at a higher rate than other organisms. Entities that are unable to cope with their environment die out, and are unable to propagate their genetic material. Thus, over a period of many generations, superior genetic material proliferates and the ability of a population of organisms to cope with its environment increases. This process is referred to as *natural selection*, or *survival of the fittest* (Koza, 1992*b*). It is interesting to note that there is neither a "driving force" behind natural evolution, nor a concept of progress: evolution is simply a result of the statistical process of organisms unable to survive in their environment disappearing from the population (Colby, 1996). Another common misconception is that the fitness of an individual in its environment determines the survival of its genes. This is only indirectly true: it is the ability of the organism to *reproduce* that determines the transferral of its genetic material (Colby, 1996).

Natural evolution would not work without variations in genetic material. The genetic encoding of each organism is contained in a very long DNA (deoxyribonucleic acid) molecule called a *chromosome*. The chromosome subsists in the concatenation of many nucleotide bases. There are four different nucleotide bases: adenine (A), cytosine (C), guanine (G) and thymine (T) (Koza, 1992*b*). The human chromosome contains about 3,100,000,000 of these bases. Thus there are $4^{3,000,000,000} \approx 10^{1.806\times10^9}$ distinct chromosomes possible, far more than the number of people who have ever lived on the planet. Every sequence of three bases is translated to one of the twenty amino acids that make up cellular life on earth via a universal mapping referred to as the *genetic code* (Altman, 1997). A cellular apparatus called a *ribosome* "reads" the DNA strand to create a sequence of amino acids. The sequence of acids forms a chain called a *protein*. Proteins have structural properties that determine the macroscopic properties of the organism; for example, there are light-receptor proteins, muscle proteins and myoglobin which stores oxygen.

Sub-strings of the chromosome containing about a thousand nucleotide bases are called *genes*. Each gene specifies a protein of about 333 amino acids. The set of genes which constitute the genetic code of an organism is called a *genotype* or *genome*. Humans have about 100,000 genes (100,000,000 DNA bases), the remaining 2,900,000,000 bases are for control information (Altman, 1997). It is an organism's genes and control information that determine its behaviour, development, disease, body structure and microscopic structure. Hence there is a duality between the genotype and the macroscopic organism, or *phenotype*.

Each gene can take on one of several values, called *alleles*. The position of a gene in the chromosome is called its *locus*.

It was at first thought that each gene in a chromosome independently controlled a single phenotypic trait. Although this view of a single gene mapping to a single phenotypic trait has been taken in the past to facilitate tractable analysis, it is not the case in general and has been labelled "beanbag genetics" (Fogel, 1995). *Pleiotropy* refers to the fact that a single gene may have an effect on multiple phenotypic characteristics. *Polygeny* refers to the fact that a single phenotypic characteristic may be controlled by the non-linear interaction of several genes. Naturally-evolved systems exhibit extensive pleiotropy and polygeny. This non-linear interaction between genes is called *epistasis*. Despite the complexity introduced by epistasis, a genotype can contain *co-adapted sets of alleles*, which are alleles occurring in polygenically-related genes that augment the fitness of the phenotype in its environment (Holland, 1995). For instance, the co-adapted alleles in fish that result in their gills make them fit to survive in their watery environment. Thus one can take a semi-modular view of genetics.

3.3.2 Evolutionary computation

Biological evolution is viewed by many as an optimisation process which proceeds by trial-and-error in a similar manner to human learning (Fogel, 1995, pg. 67). There have been many endeavours to simulate evolution on a computer in order to solve practical problems. These endeavours have resulted in the field of *evolutionary computation*. There are now four main types of algorithm based on evolutionary principles:

- genetic algorithms
- genetic programming
- evolution strategies
- evolutionary programming

All evolutionary computation methods have the following common framework:

- A population $A = \{a_i\}^M$ of M individuals is maintained, each individual $a_i \in \mathcal{A}$ being a solution to the problem.
- There is a user-defined *fitness function* $f : \mathcal{A} \rightarrow \Re$ which quantifies how well an individual solves the problem.
- The initial population, $A(0)$, is randomly generated.
- Evolution proceeds as the current population is iteratively transformed to a new population through selection, reproduction, recombination and mutation:

  $$A(g+1) = T(A(g))$$

  where $T(.)$ is the stochastic transformation process. Although T varies from algorithm to algorithm, it generally involves two stages. The first stage is the fitness-proportionate selection of individuals from the population to form the *mating pool*. The second stage is the application of genetic operators to individuals in the mating pool to produce offspring. These offspring make up the new population. One iteration of this transformation is called a *generation*.
- Evolution proceeds until some termination criterion is met. Often this is a hard limit G on the number of generations.

This process describes one *run* of the algorithm. Since the results come from a stochastic process, it is recommended that multiple runs be performed. If R runs are made, then the fitness function is evaluated $R \times G \times M$ times. For practical problems the fitness evaluation is non-trivial and dominates the computation time of the algorithm.

Evolutionary methods have the following advantages over other techniques:

- They are broadly applicable, both in the use of fitness function and the representation of solutions.
- Optimisation is based on observed values of the fitness function only: no gradient information is required.
- Learning is stochastic, so the optimisation can escape local extrema.
- The presence of a population of solutions can result in robust adaptation to non-stationary environments (Vavak and Fogarty, 1996; Calabretta *et al.*, 1997), and faster convergence than with the use of a single solution.
- The population-based approach is useful for multi-objective optimisation problems using the concept of *Pareto optimality* (Goldberg, 1989; Langdon, 1995; Fonseca and Fleming, 1995; Horn and Nafpliotis, 1993).

The main disadvantages of evolutionary computation techniques are that they:

- can be difficult to successfully apply to a problem,
- are computationally intensive, and
- can converge prematurely.

Table 3.1 summarises the characteristics of the four main evolutionary computation paradigms. Each of the evolutionary paradigms is briefly described in the following subsections, followed by detailed sections on genetic algorithms and genetic programming.

Table 3.1: The main characteristics of each of the four evolutionary computation paradigms.

Paradigm	GA	GP	ES	EP
date and first author	Holland 1975	Koza 1992	Rechenberg, Schwefel, Bienert 1964	L. Fogel 1962
solution representation	fixed-length string	tree	real-valued vector	finite-state machine
primary method of genetic variation	crossover		mutation	
selection method	fitness-proportionate		$(\mu + \lambda)$, (μ, λ)	$(\mu + \mu)$
philosophical level of evolution	genotypic		phenotypic	

3.3.3 Genetic Algorithms

The *Genetic Algorithm* (GA) is the most well-known paradigm of evolutionary computation. GAs were first proposed by John Holland (Holland, 1995, first published 1975), and a seminal treatment has been given by Goldberg (Goldberg, 1989). GAs operate on an encoding of the problem solution in an analogous manner to the manipulation of natural genetic encodings.

In the traditional GA, the solution to a problem is encoded as a fixed-length string of bits. This binary genotype is decoded by a user-defined function to produce a phenotype, which is a solution to the problem. The fitness function is applied to the phenotype, and a probability of selection is assigned to each individual with a magnitude proportional to its fitness relative to the rest of the population. Individuals are then stochastically selected according to their selection probability, and the following genetic operators are applied to obtain one offspring per individual:

reproduction: the parent individual's bits are copied exactly.

recombination: the bits from two parent individuals are stochastically combined to result in two offspring.

mutation: one or more of the parent's randomly-selected bits are flipped.

Each bit represents a gene, and its position in the string determines its meaning. The motivation for recombination of genetic material is the assumption that, for some problems, sub-sets of the chromosome determine certain traits of the phenotype independently of the rest of the string. Through re-combination, different sets of co-adapted alleles can be combined to result in an individual that is fitter than either of its parents. That is, different desirable phenotypic traits from both parents can be combined in a single offspring.

3.3.4 Genetic Programming

The *Genetic Programming* (GP) paradigm proposes the automatic generation of computer programs through simulated evolution. The ultimate goal is "what-you-want-is-what-you-get" programming: the user specifies some examples of desired behaviour, and the system generates the computer program to perform the task. GP was popularised by John Koza in his first book (Koza, 1992*b*), in which he applied GP to many problems from different domains. Genetic programming is essentially an off-shoot of genetic algorithms, with the primary difference being the structures undergoing adaptation. The individuals in a GP population are not fixed-length strings, but rather trees which can vary in size. Each tree is a parse-tree encoding of a computer program or functional expression: the internal nodes (*ie:* those nodes with children) of the tree contain functions whose arguments are the outputs from the child sub-trees, and the leaf nodes (*ie:* those nodes without children) are the inputs to the overall program. The fitness of an individual is determined by the performance of the program when run on the problem. Hence there is no decoding step as in GAs; rather the genotype *is* the phenotype. The genetic operators used in GP are analogous to GA operators, but have been modified to manipulate trees rather than strings.

The versatility of the GP representation is its strength, which enables it to solve a wide variety of symbolic problems. GP has achieved slightly superior performance than the best known human-generated solution to at least four problems in genetics and circuit synthesis (Koza *et al.*, 1996*b*). Unlike other methods, GP allows the size and structure of the final solution to be left unspecified and undergo adaptation. However, the ability of the structures to vary in size can affect search performance, and if the individuals become too large, they can hog computer resources.

3.3.5 Evolution Strategies

Evolution Strategies (ES) were jointly developed by Rechenberg, Schwefel and Bienert at the Technical University of Berlin in 1964 (Fogel, 1997). ES is used for solving continuous optimisation problems (Fogel, 1995). Each individual is a vector of real values. The initial population is generated by randomly selecting the elements of each vector in a feasible range with a uniform distribution. For each parent a_i, an offspring a'_i is generated by adding a random perturbation from a Gaussian distribution with zero mean and user-defined standard deviation to each element of the parent vector. The contents of the population in the next generation are determined by the following selection process: the parent and offspring individuals are all mixed together and ranked according to their fitnesses. The M best individuals are retained for the next generation. The process continues until some termination criterion is satisfied. Note that the real-values undergoing adaptation are meant to continuously encode phenotypic traits. Therefore small numerical mutations should bring about a small change in phenotypic behaviour.

The ES model described so far is the simplest, and is generalised by the $(\mu + \lambda)$-ES and (μ, λ)-ES (Fogel, 1995). The $(\mu + \lambda)$ model for evolution strategies uses the best μ individuals as parents to generate λ offspring. Both mutation and recombination may be involved in this process. All solutions compete for a place in the next generation. Under the (μ, λ) model, the lifetime of solutions is more limited than for $(\mu + \lambda)$, since only the μ best offspring, $1 \leq \mu < \lambda$, are selected to be the parents in the next generation. Note that while the mutation of individuals is stochastic, the selection process is deterministic (Bäck and Schwefel, 1993).

Rechenberg observed that the rate of convergence can be improved by adapting the standard deviation of mutations during the search (Fogel, 1997). The first attempt was the *one-in-five rule:* the ratio of successful mutations to all mutations should be 1/5. If it is greater, increase the standard deviation; if it is lower, reduce the standard deviation. This heuristic was obtained through experimentation on several problems. The concept of *self-adaptation* was later introduced, which attaches a co-variant Gaussian distribution to each individual for mutation, and adapts that distribution along with the solution parameters (Bäck and Schwefel, 1993). The covariance matrix $C = [c_{ij}]$ is represented by the variances $c_{ii} = \sigma_i^2$ and equivalent rotation angles α_{ij} $(\tan(2\alpha_{ij}) = 2c_{ij}/(\sigma_i^2 - \sigma_j^2))$ to ensure that C is positive-definite. These *strategy parameters* are included with the individual and are $n(n+1)/2$ in number, where n is the number of solution parameters. Each strategy parameter is also mutated along with the individual. The mutation distributions become adapted to the local topology of the fitness landscape, allowing the algorithm to negotiate a wider range of situations.

3.3.6 Evolutionary Programming

Evolutionary Programming (EP) was introduced by Lawrence Fogel in 1962 (Fogel, 1995). The aim of EP is to evolve programs that can predict their environment, and use those predictions to attain some goal. The classical framework describes the environment as a sequence of symbols from a finite alphabet. Individuals must output a symbol based on the past observed environmental states to maximise some pay-off function. Each individual in the population is a finite state machine (FSM), possessing a finite number of internal states. Each FSM responds to an input symbol from the environment by outputting a symbol from a finite alphabet; the output is based on the input symbol and the current state. Therefore, when presented with a string of input symbols, the FSM generates an equal-length string of output symbols. Individuals are evaluated based on their ability to predict future symbols from the environment. As the string of input symbols is presented to the individual, the output symbol is compared with the next input symbol. The overall fitness is based on the total prediction accuracy over the whole set of input symbols.

Individuals each produce one offspring via mutation. There are several forms of mutation: change an output symbol, change a state transition, add a state, delete a state, or change the initial state. The offspring are ranked along with their parents according to fitness, and the top M individuals are kept for the next generation; this is a $(\mu + \mu)$ selection process. An example of an early application of EP is the prediction of prime numbers. The consecutive integers starting at 1 were presented to the individuals, and each individual had to predict whether the next integer would be a prime number.

3.3.7 Philosophical Differences between GA/GP and ES/EP

The GA and GP, or *genetic*, methodologies developed almost independently of the ES and EP, or *evolutionary* methodologies. There are some fundamental philosophical differences between genetic and evolutionary methods:

- GA and GP manipulate a genetic representation of the solution, whereas ES and EP solutions are thought of as phenotypic representations. This is an important, though vague distinction. The main difference is that in GA and GP, a recombination or small mutation can result in progeny with radically different performance from the parents. In contrast, ES and EP seeks to slightly vary the phenotypic traits of the individual so that the fitness of parents and offspring are highly correlated.
- Recombination is a major feature of GA and GP, and facilitates the supposed *implicit parallelism* which is the strength of these methods (see Section 3.4.4). ES and EP, however, predominantly use mutation to obtain variation amongst the individuals.
- Genetic methods use fitness-proportionate selection to propagate better individuals at a higher rate, whereas evolutionary methods use selection as a culling force to remove unfit individuals.

The main distinction between the two philosophies is their view of what is being evolved: for the genetic methods, it is the genetic information in a population that evolves, whereas for the evolutionary methods it is the phenotypic traits that are being evolved (Fogel, 1995).

3.4 Genetic Algorithms

A flow chart for one run of the basic genetic algorithm is shown in Figure 3.2. The initial population of solutions is randomly generated, then the fitness of each individual is evaluated. The fitness function requires a user-defined mapping from the binary string to a solution. In general, the mapping is many-to-one: multiple binary strings can code for the same behaviour. Next, the mating pool of size $1 \leq M_{mp} \leq M$ is created using a stochastic selection mechanism. Although there are many different selection mechanisms, all of them select an individual in proportion to its fitness; therefore the mating pool may contain multiple copies of highly-fit individuals, but may not contain any copies of poor-performing individuals. The individuals in the mating pool undergo genetic operations to derive offspring. The three types of operator are recombination, reproduction and mutation. Recombination requires two parents and results in two offspring, while reproduction and mutation require only one parent and produce a single progeny. Each operator has a user-defined probability of application, and the operator to use is selected stochastically according to these probabilities. In the diagram, p_c is the probability of crossover, p_r is the probability of reproduction and p_m is the probability of mutation.

The manner in which offspring are inserted into the new population depends on the GA model. If the GA is *generational*, then $M_{mp} = M$ and the whole population is replaced with all the offspring. If a *steady-state* GA is used, then $M_{mp} < M$ and the offspring are placed back into the population via some replacement scheme (Jong and Sarma, 1992). The typical approach is to replace the worst individuals in the population, and commonly $M_{mp} = 1$ or 2. The steady-state GA is expected to converge faster, because offspring become available for exploitation more rapidly. Faster convergence may, however, lead to

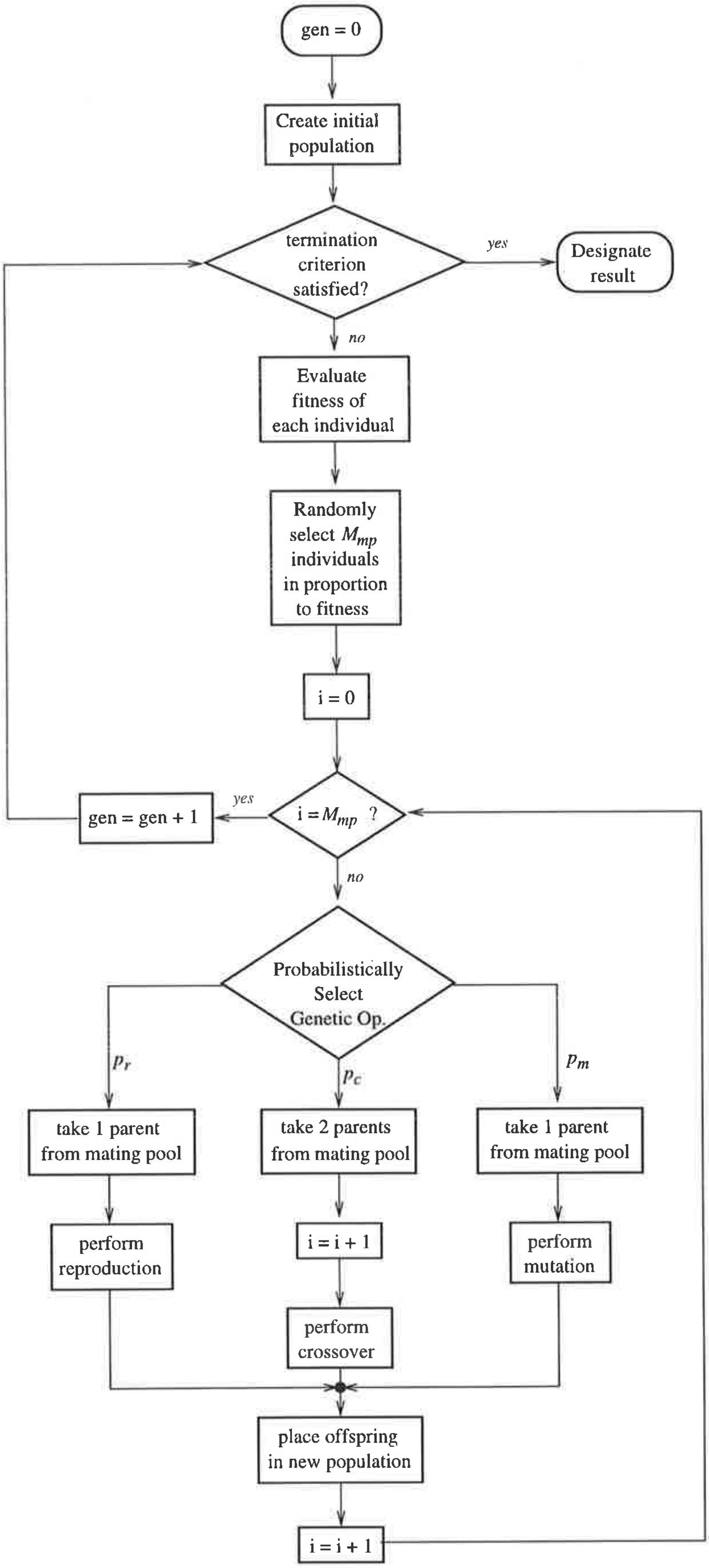

gen = 0
Create initial population
termination criterion satisfied?
yes
Designate result
no
Evaluate fitness of each individual
Randomly select M_{mp} individuals in proportion to fitness
i = 0
gen = gen + 1
yes
i = M_{mp} ?
no
Probabilistically Select Genetic Op.
p_r
p_c
p_m
take 1 parent from mating pool
take 2 parents from mating pool
take 1 parent from mating pool
perform reproduction
i = i + 1
perform mutation
perform crossover
place offspring in new population
i = i + 1

Figure 3.2: Execution flow diagram of a genetic algorithm (adapted from (Koza, 1992b, pg. 29)).

loss of diversity in the population. Since the generational model replaces all individuals each generation, it is better for promoting diversity. Other population models exist such as the *island model*, in which the population is divided up into a number of *demes*. There is a high probability of recombination between individuals from the same deme, but a lesser probability of inter-breeding between individuals from different demes. The island model is useful for avoiding premature convergence, for arriving at multiple solutions and for parallel implementation (Ryan, 1994).

It is quite possible that, upon a given generation, the best individual in the population is completely lost. To improve the quality of the final results, *elitism* is often used. Elitism simply ensures that the best individual from the previous generation is copied into the current population. In some cases, the individual is only copied if it is better than the best individual in the current population. Elitism ensures that the best-of-generation fitness is a monotonic function; if not used, the best-of-generation fitness can decrease from one generation to the next.

The specific details of the GA are discussed in detail in the following sub-sections.

3.4.1 Selection Mechanisms

The simplest method for choosing parent individuals is *roulette-wheel selection*. Each individual is assigned a probability of selection as a function of its fitness relative to the rest of the population:

$$p_i = \frac{f_i}{\sum_{j=1}^{M} f_j} \tag{3.1}$$

The selection process is then similar to a single spin of a roulette wheel in which each individual has a wedge with an included angle proportional to its probability. This analogy is demonstrated in Figure 3.3. The algorithm is implemented by randomly selecting a real number r from the interval [0.0, 1.0] with uniform probability. The selection probabilities are summed *in any order* until the aggregate reaches or exceeds r. The individual i' at which the threshold is reached is selected:

$$i' =_{i=1,\ldots,M}^{\min} \{i : r \leq \sum_{j=1}^{i} f_j\}$$

With this selection mechanism, the expected number of offspring from individual i is $M_{mp}.f_i/\sum_j f_j$.

Scaling of the fitness function before selection is common practice (Goldberg, 1989). In initial generations, there can be a wide disparity in fitness between individuals, and competition is fierce. The "tall poppies" with much higher-than-average fitness can dominate the population causing premature convergence, so in initial generations, fitness values should be scaled down. In later generations as the population converges, fitness values start to become more and more similar. To stop the search from losing its focus on the best individuals, differences in fitness need to be accentuated by scaling the scores up.

Some popular scaling schemes (Goldberg, 1989) are:

Linear: the scaled fitness values are derived from the objective scores as described in:

$$s_i = af_i + b$$

where s_i is the scaled fitness, and a and b are calculated based on the objective scores of individuals in the population. Negative objective scores are not allowed. There are many ways to choose a and b. A typical method is to maintain the same average value for the raw and scaled fitnesses, $s_{ave} = f_{ave}$, and set the maximum scaled fitness to

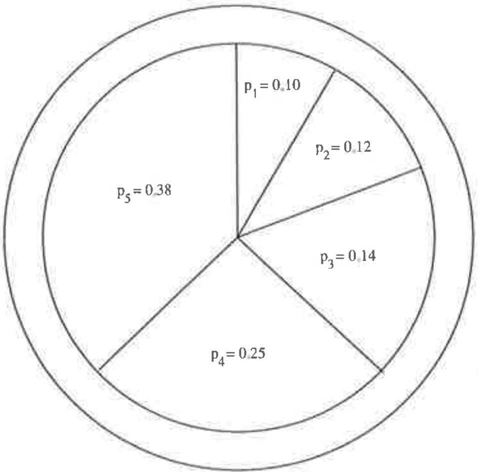


Figure 3.3: Example of roulette-wheel selection for $M = 5$, $f_1 = 10$, $f_2 = 12$, $f_3 = 14$, $f_4 = 25$, and $f_5 = 38$.

be $s_{max} = c_{mult}.f_{ave}$. Values of $c_{mult} = 1.2$ to 2 have been successfully used. This method has the advantage of being simple, but becomes problematic when the average and maximum fitnesses are close together, and the lowest fitness values are much less than the average. In such cases, the scaled fitness values can become negative.

Sigma Truncation: fitness values are scaled based on the variation from the population average. Scores can be negative, and are truncated arbitrarily at 0. The mapping from raw to scaled fitness values is:

$$s_i = f_i - (\mu_f - c.\sigma_f)$$

where c is a constant, μ_f is the mean of the population's objective scores, and σ_f is their standard deviation. Sigma truncation scaling can be used prior to linear scaling to avoid the problem of negative results.

Power Law: fitness scores are mapped to scaled values using the following relationship:

$$s_i = (f_i)^k$$

where k is a power scaling factor. The fitness values must be non-negative. The power law method has the advantage that there is only one parameter to select, which may have to be adapted during the algorithm.

To avoid the problems associated with choosing a scaling scheme and appropriate parameters, selection methods have been developed that are based only on the *relative* fitness of individuals in the population. Of these rank-based methods, the most popular is *tournament selection*. Using this scheme, a group of S individuals are selected at random from the population with uniform probability. A competition is held within this group, and the winner is the result of the selection process. The probability with which individual i is selected for addition to the mating pool is not as simple as the expression for roulette wheel selection, shown in Equation(3.1). The equivalent expression has been derived for tournament selection in (Bäck, 1994):

$$p_i = M^{-S}((M - i + 1)^S - (M - i)^S)$$

where the individuals have been sorted in increasing order of fitness (for a minimisation task): $f(a_1) \leq f(a_2) \leq \ldots \leq f(a_M)$.

Researchers often speak of *selection pressure*, which qualitatively refers to the rate at which better individuals are selected for future trials. *Take-over time* has been developed as a measure of the selective pressure of a selection mechanism (Bäck, 1994). It is defined as the number of generations required for the best-of-generation individual in the initial population to fill all positions in the population through the operation of selection only. Shorter take-over times correspond to stronger selection mechanisms. Since take-over time is defined without the use of genetic operators, it is only useful for comparisons between selection methods.

For tournament selection the take-over time is (Bäck, 1994):

$$\tau^* = \frac{\ln M + \ln(\ln M)}{\ln S}$$

For roulette-wheel selection, the take-over time is problem-dependent because the selection probabilities rely on f. Goldberg and Deb have shown that, without scaling, the take-over time for $f(x) = x^c$ is $\tau^* = (M \ln M - 1)/c$ (Goldberg *et al.*, 1991). According to these calculations, tournament selection provides stronger selection pressure than roulette-wheel selection by a factor of $\approx M$. The selective pressure can be modified for the roulette-wheel method via the scaling function. Similarly for tournament selection, selective pressure increases with tournament size S.

3.4.2 Genetic Operators

The most widely-used recombinative genetic operator in GAs is *crossover*. *Single-point crossover* requires two parent strings. Let the length of all strings in the population be L . A crossover site k is randomly chosen with uniform probability from the interval $[1, L-1]$. Two new individuals are created by swapping the sub-strings in the interval $[k+1, L]$. The process is shown in Figure 3.4.

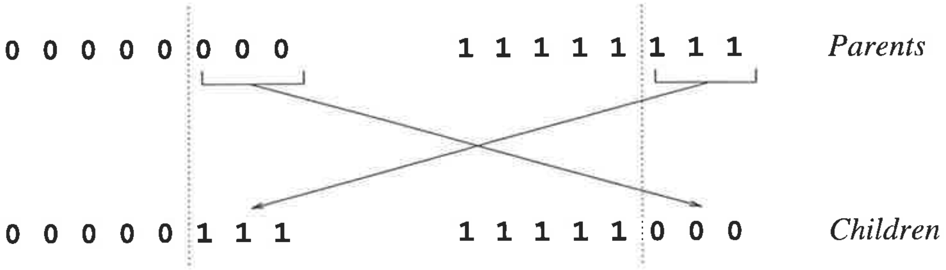


Figure 3.4: An example of single-point crossover.

Through this process, a novel solution of better utility can be synthesised from existing solutions containing desirable features, in much the same way as a human might go about combining different ideas to come up with a better idea.

The reproduction operator is very simple: it takes a single parent and just copies it verbatim to the offspring. Mutation also requires a single parent and produces a single offspring. A position is randomly selected in the individual with uniform probability, and the bit at that position is inverted. Mutation is used sparingly in GAs, and is seen as a background operator, while recombination is the driving force (Goldberg, 1989). Mutation is useful for dislodging a stagnant search, and to compensate allele loss.

The crossover operator described here is only one of many. For instance, *two-point crossover* exchanges the sub-strings between two randomly-selected crossover points. *Uniform crossover* transfers bits between individuals at randomly-selected positions rather than in contiguous chunks. The appropriate choice of operator depends on the problem and its encoding.

There is no reason why the strings should be constructed from a binary alphabet. Although the *principle of minimal alphabets* introduced by Goldberg proposed that the shortest alphabet possible should be used (Goldberg, 1989), it has since been refuted (Antonisse, 1989). In fact, anything can be placed in the elements of the strings, as long as the operators are designed to work with the representation. Also, in conventional GAs the meaning of an allele is determined by its locus. This is not always the case: for instance, *order-based* GAs such as those used for the traveling-salesman problem exploit only the relative positions of the alleles rather than their absolute positions in the string (Ronald *et al.*, 1995).

3.4.3 Diversity and Premature Convergence

Although convergence to the optimal solution is often used as a measure for an algorithm's performance, this criterion has been rejected by Holland (Holland, 1995) according to the argument that even enumerative search converges under this criterion. Rather, the best solution must be found in a "reasonable time". Convergence in genetic algorithms typically refers to the situation that the population becomes homogeneous, containing M copies of the same individual. Further search points cannot be reached through crossover, since crossing over identical strings results in identical offspring, and the very low mutation rate is the only chance of introducing new genetic material. One hopes that the algorithm has converged upon the optimal solution; if not, the only recourse is to restart the algorithm with a different initial population.

Under this definition of convergence, the diversity of genetic structures in the population is expected to decrease as evolution progresses. Due to the geometric rate at which highly-fit individuals propagate into future generations, the GA can converge too quickly without having explored enough of the search space to encounter a global or near-global optimum. This phenomenon is called *premature convergence*. In the presence of bit mutation, premature convergence is a stagnation in the search for an undetermined amount of time (Fogel, 1995).

The way to stop the GA from converging prematurely is to promote diversity in the population. There are several ways to achieve this:

- Use a high mutation rate.
- Disallow *genotypic duplicates* in the population, where two individuals are genotypic duplicates if they are exactly the same. Note the distinction from *phenotypic duplicates*, which are individuals with different strings resulting in the same behaviour.
- Use less selection pressure.
- Use a population model that promotes diversity. The island model is good for diversity preservation, since the demes are kept largely separate so that a super-fit individual cannot swamp all demes. Another single-population method is the use of *fitness sharing*, a technique for promoting speciation in the population (Goldberg, 1989). Groups of individuals exploit environmental niches, and share the resources from that niche. In practice, the fitness of each individual is de-rated by a weighted sum of the fitnesses of all the other individuals in the population:

  $$f_s(a_i) = \frac{f(a_i)}{\sum_{j=1}^{M} s(d(a_i, a_j))}$$

  where $d(a_i, a_j)$ is some measure of the distance or similarity between two individuals, and $s(.)$ is the *sharing function* that weights the distance. Individuals that are similar are given a relatively large weight, and the function drops off as individuals become less similar. As the population starts to converge, the mutually-similar individuals begin to de-rate one another's fitness values. Thus a single individual is unable to dominate the whole population without restraint.

3.4.4 Schemata Theory and Implicit Parallelism

The mathematical foundations for genetic algorithms were set out by the work of John Holland (Holland, 1995, first published in 1975). Holland's treatment attempted to characterise adaptive systems, drawing together the common threads of natural and simulated evolutionary processes. Holland defined an *adaptive plan* as an algorithm for generating new search points in light of inputs from the environment and previously-visited search points. By formalising the representation of adaptive plans, the efficiency of different plans can be compared. Holland analysed the genetic algorithm using *schemata theory*. A *schema* (plural *schemata*) is a template defining a sub-set of solutions representable by binary strings. They are formed from an augmented alphabet $\{0, 1, *\}$ where "$*$" means "don't care". For example, the schema $[0, *, 0]$ for $L = 3$ represents strings $\{[0, 0, 0], [0, 1, 0]\}$. The *order* of a schema H, $o(H)$, is the number of fixed alleles in the template. The *defining length* of a schema, $\delta(H)$, is the distance between the first and last fixed bits in the schema. For example, if $H = [1, *, 0]$ then $o(H) = 2$ and $\delta(H) = 3 - 1 = 2$.

Schemata provide a means for combining attributes, and for analysing their contribution to performance and propagation through the population. Each schema can be thought of as a hypothesis about the utility of the defined alleles. For example, the schema $[1, *, *]$ is like the hypothesis that the first bit being 1 is part of the optimal solution. The fixed bits in the schema are like "building blocks" which can be combined to form solutions. A binary string of length L has 3^L distinct schemata. Each string is a member of 2^L different schemata, and the whole population can represent at most $M.2^L$ schemata. It is natural that some schemata will be superior to others in that the average fitness of strings representing a schema can be higher. Therefore we can talk about some schemata being fitter than others, and examine the rates at which highly-fit schemata proliferate throughout the population.

Suppose that at generation g the population contains $m(H, g)$ representatives of schema H. Using the fitness-proportionate reproductive plan of roulette-wheel selection and only using the reproduction operator, the expected number of representatives of the schema in the next generation is:

$$m(H, g + 1) = m(H, g).M.\frac{f(H, g)}{\sum_{j=1}^{M} f_j(g)} \tag{3.2}$$

where $f(H, g)$ is the average *observed* fitness of schema H at generation g:

$$f(H, g) = \frac{\sum_{a_j \in A(g) \cup H} f_j(g)}{m(H, g)}$$

Noting that the average fitness at generation g is $\bar{f}(g) = \sum_{j=1}^{M} f_j(g)/M$, Equation(3.2) becomes:

$$m(H, g + 1) = m(H, g)\frac{f(H, g)}{\bar{f}(g)} \tag{3.3}$$

Qualitatively, this means that the number of instances of a schema with consistently above-average observed fitness will increase at a geometric rate with generations. It is important to note that this happens *in parallel* for all schemata in the population via simple operations on M strings. There are no calculated tables of schema representatives or schema average fitness.

Using only reproduction, it would be impossible to examine new schemata since we can only reproduce those individuals created in the initial population. To enable the examination of new search points, the variational operators of crossover and mutation must be used. Variational operators, however, modify the difference Equation(3.3) because the offspring may no longer be a member of schema H. The question is, which schemata are preserved by crossover and mutation, and which aren't? Consider the following two schemata represented by individual a:

$$\begin{array}{lcccccccc} a & = & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ H_1 & = & 0 & * & * & * & * & * & 0 \\ H_2 & = & * & * & * & 1 & 0 & * & * \end{array}$$

If the third crossover site is chosen randomly, then the individual will be divided at the point shown:

$$\begin{array}{lcccc|cccc} a & = & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ H_1 & = & 0 & * & * & * & * & * & 0 \\ H_2 & = & * & * & * & 1 & 0 & * & * \end{array}$$

Unless the second parent has the same fixed bits as H_1, the offspring will belong to H_2 but not to H_1. It is generally true that schemata with longer defining length have a higher probability of disruption through crossover. In fact, the probability of disruption of schema H is:

$$p_d = \frac{\delta(H)}{L-1}$$

The probability of survival of a schema is $p_s = 1 - p_d$, so for the example above we have $p_d(H_1) = 6/6 = 1$, and $p_d(H_2) = 1/6$. Taking into account the fact that the mating partner sometimes has identical fixed positions, the probability of survival is bounded by:

$$p_s \geq 1 - p_c.\frac{\delta(H)}{L-1}$$

Incorporating this into the difference Equation(3.3) yields:

$$m(H, g+1) \geq m(H, g)\frac{f(H, g)}{\bar{f}(g)}\left[1 - p_c.\frac{\delta(H)}{L-1}\right]$$

Now considering mutation, the probability of one of the fixed positions of schema H being mutated is $o(H)/L$. Therefore the total updated difference equation for the minimum number of schema members propagated on a single generation is:

$$m(H, g+1) \geq m(H, g)\frac{f(H, g)}{\bar{f}(g)}\left[1 - p_c.\frac{\delta(H)}{L-1} - p_m.\frac{o(H)}{L}\right]$$

This difference equation implies that short-defining-length, low-order schemata with consistently above-average observed fitness receive an exponentially-increasing number of trials in future generations. This is referred to as the *Schema Theorem*, or the *Fundamental Theorem of GAs*.

It is reasonable to question why the fitness-proportionate reproductive plan is desirable. Holland showed that the exponentially-increasing allocation of trials based on observed payoff is optimal by analogy with the *two-armed bandit problem*. A hypothetical two-armed slot machine has a different distribution of pay-offs associated with either arm, with means μ_1, μ_2 and standard deviations σ_1, σ_2. Assume that $\mu_1 \geq \mu_2$. Suppose we have N trials, or coins, to spend on the machine. The question is, what is the optimal allocation of trials between those arms? The answer is to spend all trials on the arm with the biggest expected pay-off, but we don't know which is best beforehand. If we make some initial trials on both arms, we have an estimate of the expected pay-off of each arm. If we were then to spend the rest of our money on the arm with the largest observed pay-off, there is always the chance that it was observed as best only through sampling variations, so the odd trial should still be spent on the other arm to make sure we did not go down the wrong track. There is therefore a balance between *exploiting* what we know about the relative pay-offs, and *exploring* to gain more information about the distributions of the two pay-offs. The trade-off between exploitation and exploration is a key issue in genetic algorithms: how much should we exploit the best observed schemata through reproduction, and how much should we explore schemata with low observed fitness but which may truly have a higher expected pay-off.

Holland showed that, to minimise expected loss, n^* trials should be given to the arm with the lower expected pay-off and the remaining $N - n^*$ to the better arm, where n^* is given by:

$$n^* \cong b^2 \ln \left[\frac{N^2}{8\pi b^4 \ln N^2} \right], \quad \text{where } b = \sigma_1/(\mu_1 - \mu_2). \tag{3.4}$$

Re-arranging yields:

$$N - n^* \cong N \cong \sqrt{8\pi b^4 \ln N^2}.e^{n^*/2b^2}$$

Hence the optimal number of trials allocated to the better arm should be exponentially increasing with the number given to the worse arm. Of course this strategy is unrealisable, since we do not know beforehand which arm has maximum expected returns. Holland showed, however, that the plan which allocates an initial n^* trials to each arm and then spends the remaining $N - 2n^*$ trials to the best observed arm has an expected loss that approaches the optimal loss as fast as N^{-1}.

The analogy between the bandit's arms and two schemata is surely not lost on the reader. The 2-armed bandit has been generalised to a k-armed bandit, and the optimal strategy is similar, that relatively-good performers should receive a relatively-exponential number of future trials. A k-armed bandit is analogous to a competition between k schemata with the same fixed positions but at least one different value in those positions, because these schemata compete for slots in the population.

The number of schemata contained in a population is anywhere between 2^L and $M \cdot 2^L$. Taking into account the destruction of long schemata through crossover and the possibility of multiple individuals belonging to the same schema, Holland estimated that the number of schemata processed by a population of size M is $\mathcal{O}(M^3)$. This is the theory of *implicit parallelism*, that through simple $\mathcal{O}(M)$ processing of strings, a cubic number of templates or hypotheses can be processed implicitly in parallel. Given the exponential growth rates of most search spaces, this is an attractive property.

The low-order, short-defining-length schemata processed by binary-string GAs are referred to as *building blocks*. The GA juxtaposes these building blocks like a child playing with lego, hoping that the combination of two partially-fit schemata will result in an individual of even greater fitness. This decomposition of the problem into many sub-problems allows the optimal solution to be constructed gradually rather than stumbled upon all in one step. The proposal that a GA works in this manner has been termed the *building block hypothesis*. Although the super-position of fitness is an intuitively appealing concept, its applicability to real-world problems is the important question. Naturally there are cases in which building blocks do exist for the problem, and cases where they don't. In the latter case, a change in representation or genetic operators may allow for the exploitation of building blocks.

There have been many challenges to the utility of GAs. In particular, the large random component in the algorithm superficially casts aspersions on the efficiency of GAs. Of course, in light of the power of implicit parallelism, these arguments can be refuted. The Schema Theorem and the Building Block Hypothesis, however, have themselves been called into question. Many doubt the applicability of the schema difference equation to complex dynamic systems, since it only describes the step between two generations. In (Altenberg, 1995) it is pointed out that the Schema Theorem ignores the relationship between the fitness of parents and offspring, and so is not powerful enough to describe the performance of a GA. In some research circles, the Schema Theorem is like the "Father Christmas" of GAs: something that one believes in the beginning but loses its relevance in reality. Empirical investigations have found that uniform crossover out-performs single-point crossover on a broad range of problems, even though uniform crossover does not preserve small building blocks (Syswerda, 1989).

The analogy between the k-armed bandit problem and competition between schemata is also questionable. The bandit trials are assumed independent, while trials in a GA are heavily biased towards fitter members of a schema (Grefenstette, 1991; Grefenstette and Baker, 1989). Also, in practical situations we may be interested in the schema containing the best possible individual rather than the schema with the best average fitness. Flaws have recently been revealed in Holland's analysis of the two-armed bandit problem (Macready and Wolpert, 1996). In particular, Holland's result is only optimal for the single-decision framework chosen, and does not account for the general case where a new decision is made after each trial. They point out the fact that Equation(3.4) is independent of σ_2, which casts aspersions on its correctness.

In any case, the question of whether a GA operates by processing building blocks is problem-dependent. Efforts have been made to characterise problems that are suitable for GAs by constructing epistasis measures of the representation (Mason, 1995).

3.4.5 Price's Theorem

In the field of population genetics, *Price's Covariance and Selection theorem* (Price, 1970) has been proposed to relate the change in frequency of a gene in a population to the covariance of its frequency with the number of offspring produced by individuals in the population. In recent times, the theorem has been used as an alternative to the Schema Theorem for the explanation of the operation of GAs (Altenberg, 1995) and GP (Langdon, 1996*a*).

Theorem 3.2 (Price's Covariance and Selection Theorem)

$$\Delta Q = \frac{cov(z,q)}{\bar{z}}$$

where:

ΔQ	=	*expected change in frequency of a given gene from one generation to the next*
q	=	*frequency of the gene in an individual*
z	=	*expected number of offspring produced by an individual*
$\bar{z}$	=	*mean number of offspring per individual in the population*

The theorem holds "for a single gene or for any linear combination of genes at any number of loci, holds for any sort of dominance or epistasis, for sexual or asexual reproduction, for random or non-random mating, for diploid, haploid or polyploid species, and even for imaginary species with more than two sexes" (Price, 1970). Price's theorem applies to GAs under the assumption that the genetic operators are independent of the genes.

Lee Altenberg has suggested that Price's theorem is superior to the Schema Theorem in its ability to describe how well a GA will perform (Altenberg, 1994; Altenberg, 1995). The Schema Theorem implicitly assumes a correlation between the fitness of a parent and its offspring, while Price's theorem makes this correlation explicit. Price's theorem holds for any choice of genetic operators and for any representation scheme. Altenberg has used Price's theorem to derive the Schema Theorem as a special case. The theorem implies that the search performance of a GA does not rely on schema processing, but rather on correlations between the fitness of parents and offspring.

Price's theorem has been used to predict several evolved measures in a genetic program, such as program length and gene frequency (Langdon, 1995; Langdon and Poli, 1997*a*; Langdon and Poli, 1997*b*). The theorem was validated by empirical results, except for cases in which the restriction on maximum size of programs interfered.

3.4.6 Lamarckian Evolution and the Baldwin Effect

Darwin was not the only pioneer to theorise about the evolution of species. Jean-Baptiste Lamarck (1744-1829) preceded Darwin with the concept of evolution over a long period of time. The most notable point today about his work is the hypothesis that organisms change their behaviour in response to their changing environment, and that the resulting traits are inherited over generations. A prime example is the proposal that the giraffe obtained its long neck by repeatedly stretching for the high leaves (Gaarder, 1996). The repeated use of the neck during the lifetime of the individual caused it to become strengthened, which was inherited by its offspring. It is now known that Lamark's proposal is not the case in biology, since it would require a feedback of behaviour to genetic material during the life of an organism (Waggoner, 1996).

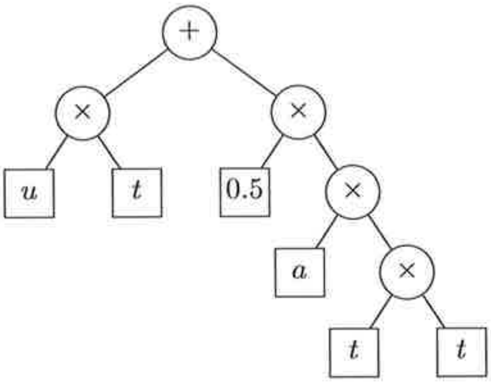
James Mark Baldwin later proposed a theory of the role of learning in evolution that could explain the apparent Lamarckian phenomenon (Baldwin, 1896). Baldwin conjectured that, although learnt traits cannot be passed on genetically, learning in individuals alters the fitness landscape to facilitate evolution. Baldwin further proposed that skills that are learned at first are replaced in later generations by genetically evolved systems. That is, abilities that require learning in one generation become innate in later generations.

Genetic algorithms and genetic programming are relatively bad at "fine-tuning" solutions to their local extremum in search space (Houck *et al.*, 1996). Therefore hybrid systems are often used in which a simple optimisation process is applied to each individual on each generation. The evolutionary algorithm performs a global search for the structure of the solution, and the optimiser performs a local search for the nearest local extremum. Lamarckian evolution and the Baldwin effect are relevant to this type of solution, because they represent the two possibilities for incorporating local optimisation information into the solutions. That is, should the locally-optimised parameters of an individual replace the original parameters, or should the genetic material remain the same and the local tuning affect the fitness function only? There is evidence to suggest that the Lamarckian approach can interfere with schema processing and lead to premature convergence (Houck *et al.*, 1996). However, the Baldwin approach aggravates the possible problem that there are multiple genotypes that map to a single phenotypic behaviour.

3.5 Genetic Programming

Genetic Programming (GP) is analogous to a genetic algorithm, except that the structures undergoing adaptation are trees rather than strings. The trees are hierarchical representations of computer programs or functional expressions. An example is shown in Figure 3.5. Under the tree is the *reverse-polish* or *prefix* notation form of the expression. The tree is said to be a *parse tree* for the expression. The internal nodes are functions that present their output to their parent node, and take the outputs of their child nodes as arguments. The leaf nodes are terminals which are the inputs to the program. The output of the program is taken from the root (top) node. The set of functions and terminals are defined by the user, and are specific to the problem. Functions can have side effects, such as altering the state of memory.

In order to apply GP to a problem, there are five things that must be specified by the user (Koza, 1992*b*):



$$+(\times(u,t), \times(0.5, \times(a, \times(t,t))))$$

Figure 3.5: An example of a genetic program to compute $s = ut + \frac{1}{2}at^2$.

- the set of terminals,
- the set of functions,
- the fitness measure,
- the parameters for controlling the run, and
- the method for designating the result and the criterion for terminating a run.

In the above example, the function set is $\mathcal{F} = \{+, \times\}$ and the terminal set is $\mathcal{T} = \{t, u, a\}$. The function and terminal sets must meet the conditions of *sufficiency* and *closure* (Koza, 1992*b*). Sufficiency requires that the set of functions and terminals be capable of expressing a solution to the problem. Closure requires each function to be able to accept as an argument any value that could possibly be output by a function or a terminal. For example, the arithmetic division function must be *protected* against division by zero:

$$\div(x,y) = \begin{cases} 1 & \text{if } y = 0; \\ x/y & \text{otherwise} \end{cases}$$

As will be shown in Section 4.10, the number of possible trees, and therefore the size of the search space, increases as a very-high-degree polynomial function of the number of functions and terminals. Therefore the function and terminal set should be limited in size for an effective search. An issue here is the level of complexity of the function set. The best results will be obtained by using as much prior information possible so that the GP does not have to re-invent the wheel. For example, on a problem involving time series it may be more sensible to use the Fourier transform as a function rather than $\{+, \times, \exp\}$, because if the Fourier transform is required, it is unlikely that the GP would synthesise it given the current limitations on computer memory and speed.

The fitness measure is obtained by executing an individual and seeing how well it solves the problem. This may require a single execution, or require the evaluation of several *fitness cases*. For example, when using GP for regression it is not practical to evaluate a program's prediction at every point in the target function domain. Therefore the fitness function can be based on a sub-set of all possible program inputs. The function set may also involve loops, so the fitness evaluation may have to terminate execution of a non-halting program.

It was stated earlier that for GAs the fitness function evaluation dominates execution time of the algorithm. There is an extra complication in GP: the number of nodes in a tree grows exponentially with depth, so large programs can take a long time just to evaluate, and they use up a lot of memory. For this reason, the maximum depth of trees is restricted during creation and generation of offspring.

The user-defined parameters required for the standard GP algorithm are listed in Table 3.2. The result of a run is usually the individual with the maximum fitness value encountered. A run is usually terminated after a set number of generations.

Table 3.2: Control Parameters required for standard GP.

population size, M
max. number of generations, G
probability of crossover, p_c
probability of mutation, p_m
probability of reproduction, p_r
probability of choosing internal points for crossover, p_{ip}
selection scheme
fitness scaling scheme
max. depth of trees created during a run, D_c
max. depth of initial random trees, D_i
method for generating initial population
elitist strategy?

3.5.1 The Initial Population

The initial population is randomly generated from the set of functions and terminals available. Since it is the starting point for the search, it is important that a range of tree shapes and sizes be present to avoid bias. Although functions and terminals absent from the population can be re-generated by mutation, this process should not be relied upon, and the initial population should contain an even distribution of the functions and terminals available. Genotypic duplicates should not be allowed so as to use the full potential of the population (Koza, 1992*b*).

Three methods for the random generation of trees are suggested in (Koza, 1992*b*): the *grow* method, the *full* method, and the *ramped half-and-half* method. The grow method randomly selects the next child node from the union of the function and terminal sets with uniform probability, so the trees can have irregular shapes. The full method only selects child nodes from the function set to give trees for which every path from the root node to a terminal node has the same length. In both cases, the user imposes a maximum allowable depth for the generated trees. When this depth is reached, nodes can only be selected from the terminal set.

The ramped half-and-half method creates equal numbers of trees at depths in the range $[2, D_i]$. At each depth, half of the trees are created using the grow method and the other half using the full method. This method is the most widely used in GP applications.

3.5.2 Genetic Operators

The traditional GP operators are analogous to those used in GAs, but have been adapted to work with trees. Each operator must ensure that the offspring do not exceed the maximum allowable depth D_c so that the trees cannot grow without bound. Reproduction and crossover are viewed as the primary genetic operators, while mutation and other novel operators are secondary and are used sparingly.

Reproduction simply copies the individual to obtain an identical offspring. Crossover requires two parents and randomly selects a node in each as a crossover point. The sub-trees rooted at these crossover points are swapped to obtain the offspring. An example is shown in Figure 3.6. There are a few important points about crossover in GP. Firstly, unlike genetic algorithms, crossover between two identical parents can result in different offspring. Secondly, if the crossover points are the roots of both parents, then crossover degenerates to reproduction. Thirdly, if the two crossover points are terminal nodes, crossover is like point mutation that simply changes a single node. Because of this latter point, internal points are typically chosen as crossover sites with a higher probability p_{ip} than terminals; Koza used $p_{ip} = 0.9$.

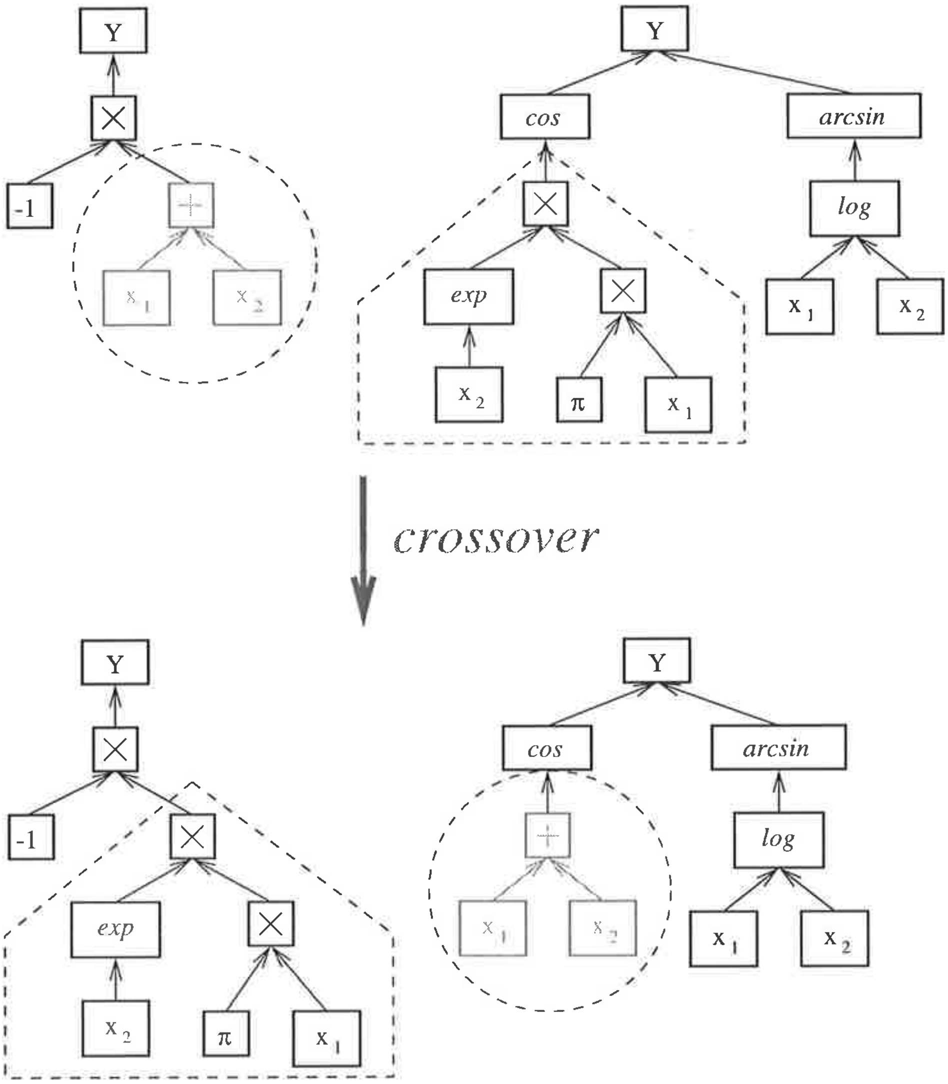


Figure 3.6: An example of sub-tree crossover.

The most common form of mutation is *sub-tree* or *grow* mutation. A node is randomly selected from the tree, and the sub-tree rooted at the selected node is replaced with a new randomly-generated sub-tree. An example is shown in Figure 3.7. The usefulness of mutation is questionable according to Koza: mutation is useful in GAs to relieve a search from stagnation and to re-introduce lost alleles into the population. Since in GP identical parents can result in different offspring through crossover, mutation is not required to avoid stagnation. In GP the meaning of an allele is not fixed to its position, and since the sizes of the function and terminal sets are much smaller than the number of nodes in the population, it is unlikely that a symbol would disappear from the population.

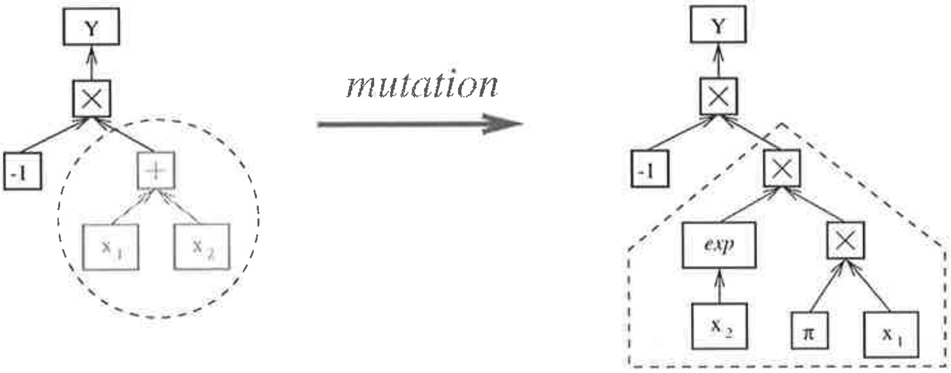


Figure 3.7: An example of sub-tree mutation.

Koza used several other secondary operators particular to GP. *Permutation* is a generalisation of the GA inversion operator. It re-arranges the order of function arguments. The operator proceeds by selecting an individual the same way as for crossover and reproduction. Next, a function in the tree is randomly selected with a uniform probability. If the function has n arguments, then one of the $n!$ possible permutations is selected with uniform probability and the arguments are rearranged to assume that permutation.

The *editing* operator visits each individual in the population and recursively applies domain-independent and domain-specific editing rules to each function. The domain-independent rule is: if any function has constant terminals as arguments, that function is evaluated and replaced by a constant equal to the result of the evaluation. Domain-specific rules may include the removal of identities or redundancies, such as replacing $(x - x)$ with 0. Editing can be performed as a cosmetic operation at the end of the evolutionary run, or as a simplifying operation with some generational frequency while the genetic program is running. The consequences of using editing during a run are unclear. On one hand, simplifying verbose sub-expressions can reduce their vulnerability to crossover. On the other hand, editing can prematurely reduce the number of available structures for recombination.

Encapsulation identifies a potentially useful sub-tree and gives it a name for future use as a function. Encapsulation allows the GP to discover its own useful building blocks. Encapsulation is demonstrated in Figure 3.8. The operator selects an individual according to fitness, randomly chooses an internal function in the tree and then prunes the sub-tree below and including that point. The sub-tree is given a function name, $E0, E1, E2, ...$, which replaces the original sub-tree at the pruning point. The function, which takes no arguments, is then added to the GP's repertoire of functions for use during mutation operations.

3.5.3 Strong Typing

The traditional form of GP generates *weakly-typed* solutions: as long as the function and terminal sets have the closure property, no distinction is made about the sensibility of a given function taking the output of another function or terminal as an argument. It is common, however, to want to combine different data types in the program trees with restrictions on how types can interact. *Strongly-typed genetic programming* (STGP) was introduced in (Montana, 1993) to ensure the correct combination of different types. The types of terminals, functions and their arguments must be specified beforehand by the user, as well

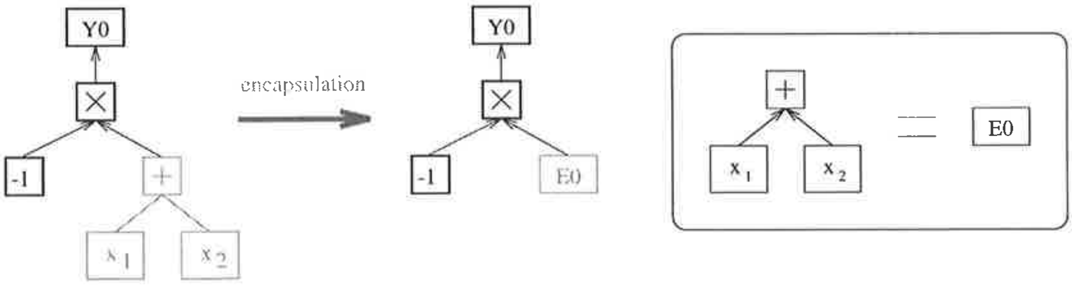


Figure 3.8: An example of the encapsulation operator.

as the return type at the root of the tree. The two constraints imposed by STGP are:

1. the root node must return a value with the type specified for the problem, and
2. each non-root node must return a value of the type required by the parent node as an argument.

The creation and genetic operators must ensure that legal parse trees result. STGP effectively reduces the size of the search space by disallowing nonsensical solutions.

One of the main complications of STGP is implementing functions which can take or return multiple types. For instance, the If-then-else function can have any type of second and third argument, as long as they are the same. Montana approached this problem using *generic functions* which are instantiated during program tree generation. Once instantiated, the program acts just like a normal strongly-typed function.

The only operator that is significantly complicated by strong typing is the program creator. Clearly the next node to be inserted below some function is restricted by the legal types allowed for that argument. A further and more subtle restriction is that the element chosen must make it possible to construct a sub-tree whose depth is within the legal limit. Consider, for example, a problem in which the function set is $\mathcal{F} = \{\texttt{OR}, \texttt{NOT}, <\}$ and all the terminals are of type Real. During construction of the tree shown in Figure 3.9 with maximum depth 3, no node of return type Boolean can be placed at "?" without violating the depth restriction, because < is the only function available that returns a Boolean result.

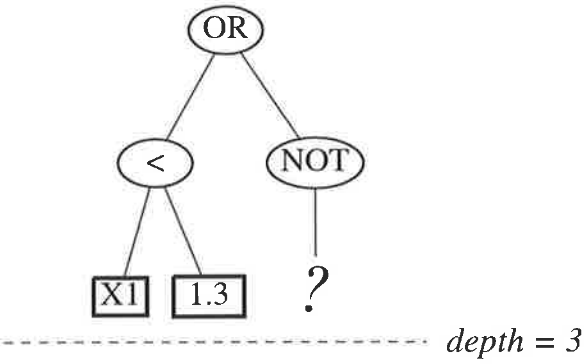


Figure 3.9: Example of an illegal type choice during program generation.

This problem is overcome by generating off-line a *node possibility table*[1], which contains for $i = 1, \ldots, D$ the possible functions or terminals for the root node of a tree of maximum

[1] In Montana's work, a *type possibility table* is used which contains the legal return types for the given depth. In general, however, a function can be legal or illegal regardless of its return type. In Figure 3.9, for example, the `NOT` node is illegal, but a `<` node at the same depth is legal, and they both return Boolean.

depth i. There must be one table for the grow method and one for the full method. The pseudo-code for generating these tables is shown in Figure 3.10.

```
// trees of depth 1 must be a single terminal
for each element j in the terminal set, loop:
   add element j to grow_table(1) and full_table(1);
end loop;

for each remaining depth i = 1 to D, loop:
   // for the grow method, legal trees of size i-1
   // are also legal trees of size i
   add all elements from grow_table(i-1) to grow_table(i);

   for each non-terminal element j, loop:
      if each of the arguments of j is matched by the return
      type of at least one element in grow_table(i-1), then
      add j to grow_table(i);

      if each of the arguments of j is matched by the return
      type of at least one element in full_table(i-1), then
      add j to full_table(i);
   end loop;
end loop;
```

Figure 3.10: Pseudo-code for generating the node possibility tables.

3.5.4 Validity of the Building Block Hypothesis

Since the algorithmic framework for GP has been borrowed by analogy from GAs, it has been taken for granted that the building-block hypothesis also exists for GP. Useful sub-trees have been found ubiquitously in the population for some problems (Koza, 1992*b*; Tackett, 1994; Rosca and Ballard, 1994). The existence of a schema theory for GP has, however, been questioned very critically in the literature. The first question to ask is: what is a schema in GP? Several attempts have been made at defining a general template for variable-sized programs (Koza, 1992*b*; O'Reilly and Oppacher, 1995*b*; Whigham, 1995; Haynes, 1997; Poli and Langdon, 1997). All of these definitions are complicated by the variable size and structure of GP individuals. Under their framework, Poli and Langdon developed a GP schema theorem for a specialised crossover operator. The theorem is more complicated than that for GAs. They predict that using this operator, the GP conducts the search in two continuously-joined phases: first a search for the optimal structure, and then a search for the optimal contents of that structure.

If the building-block hypothesis was tenuous for GAs, then it is more so for genetic programming because the functions used are typically non-linear. For example, the usefulness of the expression $x < c$ can be discontinuously reliant on the value of c. Therefore under any definition of a schema, one would expect the variance of a schema to be very large. The assumption that a functional expression can contribute to fitness in almost any context is applicable in few instances. In (O'Reilly and Oppacher, 1992; O'Reilly and Oppacher, 1994*c*; O'Reilly and Oppacher, 1995*b*) the issues involved in a GP building-block hypothesis are explored. The quantities from the GA Schema Theorem that are assumed to remain relatively fixed over generations, namely relative observed schema fitness and probabilities of destructive crossover and mutation, cannot be assumed to be so stable in GP since the defining-length of a schema can change from one generation to the next.

Empirical evidence has shown that the mutation operator, which has traditionally been thought to be a background operator in GP, can outperform crossover. In (Angeline, 1997*b*), a comparison was made between the standard GP using crossover and the use of "headless-chicken crossover", which replaces one parent with a randomly-generated tree before recombination. The results showed that headless-chicken crossover performed as well as or better than standard crossover. Angeline's explanation was that, rather than processing building blocks with above-average fitness, the GP was progressing by means of mutation. Standard crossover exhibited inferior performance because the pool of random sub-trees was limited to those taken from the current population. Another comparison between crossover and mutation (Luke and Spector, 1997) found that, although crossover gave better performance on average, the improvement was rarely statistically significant, and that mutation often performed better than crossover using small populations. Another comparison between standard GP and GP without crossover but with a set of different mutation operators has shown favourable performance for the mutation operators (Chellapilla, 1997).

Overall the evidence implies that the standard single-point crossover operator used in GP does not exploit building blocks in all situations, if they exist at all. The philosophy of Angeline and Chellapilla seems most appealing: that different, complementary operators should be used when the optimal operator for a problem is not known beforehand. A clever approach by (Teller, 1996) to finding the optimal operators for a given problem is to co-evolve a population of recombination operators with the population of solutions. A population of operators is maintained that are able to combine parent solutions in an arbitrary manner. These operators are used to modify the individuals in the population of solutions. A fitness value is then assigned to each operator based on the average improvement in fitness it brought about on the solutions. The operators then undergo adaptation, and repeating this process they evolve along with the population of solutions.

3.5.5 Automatically-Defined Functions

The functions and terminals supplied by the user are primitive components from which the GP solutions are constructed. In programming code written by humans, a problem is subdivided into sub-routines which can be invoked multiple times. Similarly if the building block hypothesis applies to a certain GP problem, then generally-useful macro-functions constructed from the primitives should be found throughout the population. The next logical step is to incorporate such macros into the function and terminal sets to facilitate a more powerful search. This is a significant concept, as it allows the algorithm to accumulate knowledge about the problem, perhaps even knowledge that would be applicable in other domains.

The encapsulation operator described earlier is an example of a scheme to accumulate sub-routines. There have been several other investigations into this concept. *Hierarchical genetic programming* (Rosca, 1995*b*; Rosca, 1995*a*) adapts the GP representation through the discovery of new useful macro-functions. Another system constructs a library of functions during evolution (Angeline and Pollack, 1993). The most popular method is the topic of Koza's second book, *automatically-defined functions* (ADFs) (Koza, 1994*b*). Using the ADF representation, each individual has a set number of sub-routines and a main program. The main program can call each of the sub-routines any number of times. The main thrust of the work on ADFs is in *exploiting regularity*. Many real-world problems of interest contain repetitive structure. ADFs are able to exploit this by repetitively applying a useful sub-routine. Alternatively, if the sub-routines were to be inserted verbatim into the main program, the size of the resulting program would be unmanageable. ADFs have been applied to classification of gene sequence patterns (Koza, 1994*a*) and analog circuit synthesis (Koza *et al.*, 1996*a*).

3.5.6 The Bloating Phenomenon and Parsimony Pressure

A phenomenon that has been observed by many GP practitioners is the tendency for the average size of trees to increase with generations (Blickle and Thiele, 1994; Nordin and Banzhaf, 1995; Nordin *et al.*, 1995; Tackett, 1994). Since the initial population is usually biased towards smaller trees, it is natural that the population will grow to the appropriate size as accuracy is gained. However, the code growth continues even when no improvement in fitness is made (Blickle, 1996*a*). This phenomenon has been termed *bloating*, or the *size problem*. Bloating is undesirable because larger solutions are difficult to read, tend to generalise poorly (Zhang and Mühlenbein, 1995), and consume more system resources. It has been suggested that bloating is useful in non-stationary environments, and generally facilitates evolution because it can protect solutions against destructive crossover (Langdon and Poli, 1997*b*), as we shall see in a moment.

The three ingredients for bloating are *selection pressure*, a *variable-length representation* and *representational redundancy*. In GP it is generally the case that the same behaviour can be represented by many different trees. It is also generally true that the fitness of the best encountered solution increases quickly with initial generations, but improvements are more difficult to make later in a run (Langdon and Poli, 1997*b*). Langdon observed that most crossovers in later generations result in no change in fitness, so the evolution degenerates to a random search for ways to represent the same behaviour. Since there are many more ways to represent the same behaviour with larger solutions, more of these become prevalent.

Analysis in (Blickle and Thiele, 1994) focused on redundant portions of GP trees. An edge A in a tree T was defined as redundant if, for all values of the terminals (inputs) of T, the function represented by T is independent of the sub-tree of edge A. That is, the fitness of a program is independent of a redundant sub-tree. Often this occurs because a sub-tree is never executed in the fitness calculation; for example, in the expression If($True$, T1, T2) the expression T2 is never evaluated. Redundant code is generally more subtle than this example, involving portions that look useful but are never executed for the particular fitness cases used. These redundant segments of code are also referred to as *implicitly-defined introns* (IDIs). Qualitatively, the argument of the analysis in (Blickle and Thiele, 1994) is that when the search comes to an end and no improvements in fitness can be made, the individuals that procreate the most are those whose fitness is not degraded by crossover. Given that there is an effective portion of a tree and a redundant portion, the fitness will decrease if a crossover or mutation point is selected within the effective portion. Therefore trees with a larger redundant portion are less likely to undergo destructive crossover or mutation because the redundant code acts as a decoy, protecting the effective code. Blickle and Thiele showed that trees which contain more redundancy, and are hence larger, are given an exponentially-increasing number of future trials relative to smaller trees. Thus the population grows without bound.

There are three types of method for controlling code growth in GP (Langdon and Poli, 1997*b*):

Restrict Size: Restrict the maximum depth or maximum number of nodes in each tree. This is really a band-aid solution because the size of the optimal solution is not known beforehand and the restriction may stop the solution from being found.

Modify Fitness Function: The fitness function can be augmented with a complexity penalty:

$$f'(a) = f(a) + \alpha.C(a)$$

where $C(a)$ is some measure of the complexity of an individual, such as the number of nodes, and α is a constant. The problem is in the selection of α. The minimum description length principle has been applied to GP for regression (Iba *et al.*, 1994*a*). The fitness function becomes the sum of the encoded length of the tree and the encoded length of the exceptional fitness cases that the tree got wrong:

$$f(a) = \underbrace{0.5k \log N}_{\text{tree coding length}} + \underbrace{0.5N \log S_N^2}_{\text{exception coding length}}$$

where N is the number of fitness cases, S_N^2 is the mean square error on the cases, and k is the number of parameters in the particular tree representation used.

Adaptive methods have also been proposed. The *adaptive Occam method* described in (Zhang and Mühlenbein, 1995) uses a two-stage process of learning in which parsimony pressure is increased as the classification error decreases:

$$\alpha(g) = \begin{cases} \frac{1}{n_{val}^2} \frac{E_{best}(g-1)}{\hat{C}_{best}(g)} & \text{if } E_{best}(g-1) > E_{max} \\ \frac{1}{n_{val}^2} \frac{1}{E_{best}(g-1).\hat{C}_{best}(g)} & \text{otherwise} \end{cases}$$

where $E_{best}(g-1)$ is the fractional error of the best individual from the previous generation over n_{val} fitness cases, E_{max} is the maximum allowed error for solutions generated using this method, and $\hat{C}_{best}(g)$ is an estimation of the complexity of the best individual in the current generation, calculated using a moving average:

$$\begin{aligned} \Delta C_{sum}(g) &= \frac{1}{2}[C_{best}(g) - C_{best}(g-1) + \Delta C_{sum}(g-1)] \\ \hat{C}_{best}(g+1) &= C_{best}(g) + \Delta C_{sum}(g) \end{aligned}$$

When the best-of-generation error is greater than E_{max}, parsimony is of little consequence since we want to grow the trees to obtain the desired performance. Once the error drops below E_{max}, $\alpha(g)$ increases to force the size of the trees down, while maintaining the same accuracy. This method is troublesome, however, because the targeted maximum error is not always known. A similar scheme is investigated in (Blickle, 1996*b*), where the fitness function being minimised switches discretely to tree complexity when the desired accuracy is reached.

These methods are all a basic form of multi-objective optimisation. Multi-objective methods have explicitly been used to encourage parsimony with the concept of Pareto optimality (Langdon, 1996*b*). In the work of (Ryan, 1994), accurate programs and small programs were bred independently with controlled rates of inter-breeding.

Modify Genetic Operators: Crossover and mutation can be biased towards smaller trees, or made aware of redundant code. (Blickle and Thiele, 1994) used *marking crossover* by identifying those tree edges[2] not traversed during fitness evaluation and avoiding crossover at the marked edges.

Methods for biasing the GP towards smaller offspring are called *parsimony pressure.* Given the cause of bloating, it seems that the logical method is to remove all redundancy

[2] An edge in a tree denotes the link between two nodes.

from the representation. Unfortunately this is only possible in special cases because removal of all redundant code is reducible to the program equivalence problem, which is nonrecursive (Soule *et al.*, 1996). Removal of redundancy was examined in (Soule *et al.*, 1996), but they were unable to rid the programs of all IDIs and the programs eventually began to grow. An example of a case where redundancy can be removed is in boolean functions (Droste, 1997). Any boolean function can be represented by a directed acyclic graph called an *ordered boolean decision diagram* (OBDD). The reduced form of an OBDD has the important property that it is unique for the given functional behaviour.

3.6 Conclusion

This chapter has presented a brief introduction to the field of evolutionary computation, and in particular genetic algorithms and genetic programming. The next chapter starts the novel content of this thesis by motivating a search over pre-processors for supervised classification. Chapter 5 describes how genetic programming can be used to that end. Experimental results of the application of GP to pattern recognition are presented in Chapter 6, and the conclusions are then given in Chapter 7.

Chapter 4

A Framework for Automatic Feature Extraction

4.1 Introduction

The reader has been introduced to the fields of pattern recognition in Chapter 2 and of evolutionary search techniques in Chapter 3. This chapter presents the main ideas and motivations for the research of this thesis. In brief, the main idea is that of automatic, or generalised, feature extraction. It has been noted that there is no existing formalism or framework for feature extraction. A framework is suggested here which is a more general form of existing pre-processing methods.

First, existing mechanisms for feature extraction are reviewed and their shortcomings pointed out. Then a series of stand-points, or hypotheses, are presented along with their justification. These logical steps in thought lead to a *generalised pre-processor*, one which is not only universally applicable but also realisable with finite resources. Next, the issue of how to synthesise such a pre-processor is raised, and population-based heuristic search techniques are suggested. Two questions are posed concerning the feasibility of synthesis and practical use of these pre-processors, which lead to the construction of the evolutionary pre-processor described in Chapter 5 and the empirical study detailed in Chapter 6.

4.2 Manual Feature Extraction

It was pointed out in Chapter 2 that there are many *general* classifiers which objectively treat the input data as points distributed in a high-dimensional feature space. These classifiers are general because no knowledge of the problem is required to apply them, save that the assumptions made by the classification algorithm should be concomitant with the data. The step that makes these methods successful in practice is the *pre-processing* or *feature extraction* stage, the transformation applied to the raw data before classification. Pre-processing is required to reduce the dimensionality of the data, and to improve the discriminatory power of the inputs to the classifier. Note that dimensionality reduction is beneficial by alleviating the curse of dimensionality and reducing computational requirements. Features may also be extracted that are invariant to transformations such as scaling or rotation. While the boundary between pre-processor and classifier is in some cases indeterminate, the distinction between the two stages can generally be made as follows: feature extraction is problem-dependent, but classification is more general. For this reason, one finds many broadly-applicable classifiers, such as k-Nearest Neighbours clustering, Perceptrons and Parzen Windowing, but few generally-useful pre-processors.

Data are usually pre-processed in a way that is chosen by the operator and is specific to the problem; this practice will be referred to as *manual feature extraction.* There is little theory to guide the feature-extraction process, and the operator typically draws on domain knowledge, past experience, trial-and-error and intuitive ideas about which features are important (Nilsson, 1993; Fu, 1980). There are many examples of ad-hoc manual feature extraction: a few are given here.

An example of manual feature extraction comes from the technique for recognition of human faces described in (Dunstone, 1995). The facial images were first normalised using low-frequency Gabor transformations, then local features were sought using Gabor functions. The motivation for using Gabor functions is biological, as they have been found in the mammalian visual system. Another example is found in (Casasent and Neiberg, 1995), where distortion-invariant recognition of objects from infra-red images is performed. Wedge-sampled magnitude-squared Fourier transforms were used as features because the designer knew from experience that they would be appropriate for the application.

A common approach to feature extraction is to synthesise a large set of features and select the most effective sub-set of these. The Multi-function Target Acquisition Processor ATR System used in (Tackett, 1994) and the **segment** data set used later in this thesis (see Appendix C) are both examples of the extraction of many (26 and 19 respectively) localised non-linear features from images. In (Breiman *et al.*, 1984) a section on the use of decision trees for selection of a sub-set of features reads (bold face added):

> Following this **intuitive** appraisal, 55 new features were constructed. These were averages, $\bar{x}_{m_1,m_2}$ over the variables from m_1 to m_2 for m_1, m_2 odd, that is
>
> $$\bar{x}_{m_1,m_2} = \frac{1}{m_2 - m_1 + 1} \sum_{m=m_1}^{m_2} x_m, \; m_2 > m_1$$

The selection of appropriate features for classification by a practitioner must be performed with care, because the classifier *cannot* compensate for a poor choice of features (Nilsson, 1993). Manual feature selection is problematic for the following reasons:

- There is very little theory to guide the selection, except for the designer's intuitive ideas about which features are important (Nilsson, 1993; Fu, 1980).

- For a difficult problem, there may be no intuitive understanding of the data. The measurements provided may come from disparate sources, and may be intuitively difficult to combine. The scale-factor chosen for each measurement can greatly affect the performance of the classifier (Duda and Hart, 1973).

- The best number of features is not known. Nevertheless there are methods for estimating the *intrinsic dimensionality* of a data set, which is the dimensionality of the smallest non-linear sub-space which entirely contains the data (Bishop, 1995). One method estimates the fractal dimension of the data as a measure of self-similarity to determine the necessary number of features (Aviles-Cruz *et al.*, 1995). The difficulty with these methods is that they may not necessarily produce the best number of features *for use with the classifier.*

- Selecting features by trial-and-error, which may involve some dimensionality-reduction work to visualise the clusters in the data, can be time consuming and laborious.

- Pre-processing is usually developed independently of the classifier, whereas the features and the classifier are inter-dependent.

- There is no concept of optimality. Although there is no problem with using intuition and experience per se, there is no way to be sure that the operator has done the best job possible in manually extracting the features.

Optimal features must be optimal with respect to some criterion, and the selection of an appropriate criterion for a pre-processor is not obvious. For a classifier, the objective is clear: to minimise the error rate. The criterion for good features suggested here is to minimise the error rate of the classifier *on the pre-processed data.*

4.3 Automatic Feature Extraction

Given the problems associated with manual feature extraction, a general framework is required under which the pre-processing stage can be optimised. One may question the possibility of a general pre-processor, since it has been stated in this thesis that pre-processing is the problem-dependent component of pattern recognition. Consider, however, the following pattern recognition tasks:

1. understanding of human speech,
2. recognition of human faces,
3. hand-written character recognition,
4. segmentation and classification of occluded objects in a visual scene, and
5. medical diagnosis via evaluation of diagnostic evidence.

These are all quite different tasks with different subtleties and data types, but they have something in common: these tasks can all be performed by humans using neurons and adaptive learning processes. The implication is that there are common aspects to these problems which can be addressed under a general framework: an exemplary system being the human brain. The extraction of features in a problem-independent manner will be referred to here as *automatic feature extraction.*

There are existing methods that extract features in a problem-independent way. The most common is *principal component analysis* (PCA), which projects the data onto the principal axes of maximum variance in the data. The projections onto those axes with the largest variance are used as features. There are two problems with PCA. First of all, the features are linear combinations of the inputs and are not powerful enough to describe non-linear data (Bishop, 1995, pg. 314). Secondly, PCA is useful for *description* of data, but not always for *discrimination.* In some situations the lower-variance principal components may contain the discriminatory information in the data, but would be discarded by PCA (Bishop, 1995, pg. 318). Discriminant analysis provides methods for extracting linear features with discriminatory qualities. These methods find the optimal linear transformation of the data according to a class-separability criterion (Fukunaga, 1990). Unfortunately, one needs to know something about the structure of the naturally-occurring clusters, since different criteria are appropriate for different cluster configurations (Duda and Hart, 1973).

Some people in pattern recognition research have attempted to build a perfect classifier and de-emphasise feature extraction. In such cases the pre-processor is intrinsically contained in the classifier. For example, a multi-layer perceptron used for classification is often fed the raw data at its inputs with no explicit pre-processing. The non-linear hidden units transform the input data in successive stages, with a different representation from one layer to the next. It has been observed in (Gallinari *et al.*, 1991) that for these internal representations, the clusters in the data are progressively more separated and compact from one layer to the next. Thus the early layers of the network learn to perform feature extraction, and the final layers perform classification.

The MLP suffers from two drawbacks which detract from its generality but are different from the limitations of PCA. Firstly, the activation functions used in the hidden nodes are sometimes inappropriate for a given problem. Secondly, the optimal structure and interconnection for an implemented network is unknown beforehand. These problems will be discussed in detail later, but it must be noted that both arise from a discrepancy between the theoretical existence of a network to perform the required mapping promised in Theorem 2.1, and the limitations of what is practically realisable on current-day computers.

Therefore the MLP could not, for instance, be used for classification of 1000 × 1000 natural images because this would require over n million weights in the first hidden layer for n hidden nodes, and the data would be too sparse for the optimisation of so many weights. The computational and dimensional problems of the fully-connected MLP have been overcome by *convolutional networks* (Lawrence *et al.*, 1996) through the use of three mechanisms: local receptive fields, shared weights and spatial sub-sampling. Each layer in the network contains neurons that are connected to only a sub-set of the units in the previous layer, and all neurons in a layer have the same weight values.

Another automatic method for non-linear feature extraction is to use an auto-associative network which simply learns the mapping from each training instance to itself, but which has a hidden layer in the centre with only a few hidden nodes (Bishop, 1995; Haykin, 1994). The output values at the hidden nodes for a given input constitute a low-dimensional non-linear representation of the data. This technique requires at least three hidden layers and expensive training algorithms.

The methods for automatic feature extraction listed in this section can be considered as frameworks for generalised pre-processing, albeit unwitting ones. In general, all existing methods for automatic feature extraction are restricted to some fixed structure and functions of which the features are composed. This imposes a limitation on the applicability of such techniques and on the range of underlying physical generative processes that can be efficiently modeled.

4.4 Contributions of this Thesis

This thesis introduces the concept of a *generalised pre-processor* (GPP) which is not found explicitly in the literature. The generalised pre-processor provides a framework for automatic feature extraction under which optimisation of the pre-processing stage can be performed. The GPP extends the traditional Universal Approximator to be both universally applicable *and* implementable using finite resources. The optimal GPP must be sought in the context of the classifier used. The shift of focus towards common facets of pre-processing rather than classification is in itself a contribution to pattern recognition research, and may become the launching pad for future fundamental developments in the field. Nevertheless, the hypotheses and investigative questions introduced in this sub-section constitute the formal contribution of this thesis.

Before proceeding to the hypotheses and research questions, let us define a *constituent function* to be any function from which a pre-processor is composed. Note the distinction between constituent functions and *basis functions*. The term "basis function" traditionally refers to a function appearing in a linear combination. For example, the sine and cosine functions are basis functions for the Fourier series representation. The definition of constituent functions is more general than that of basis functions. For example, the addition and multiplication functions are also included as constituent functions of the Fourier series representation. A distinction is commonly made between a *linear* function and a *non-linear* function. A linear function is a linear combination of its arguments, *ie:* $f(x_1, x_2, \ldots, x_d) = a_0 + a_1.x_1 + a_2.x_2 + \ldots + a_d.x_d$ where the a_i are constants. A non-linear function is any other non-constant function.

The usefulness of a GPP is motivated by the following three conjectures:

Hypothesis 4.1 *A pre-processor constructed from the appropriate constituent functions can be more economically realisable than existing methods of universal function approximation.*

Hypothesis 4.2 *A pre-processor having the appropriate structure can be more economically realisable than existing methods of universal function approximation.*

Assuming that the results of these two hypotheses are positive, it follows that:

Hypothesis 4.3 *A pre-processor having the appropriate structure and constructed from the appropriate constituent functions can be more successfully applied than existing methods of universal function approximation.*

On the basis of these hypotheses, one would like to construct a pre-processor using arbitrary non-linear functions and having arbitrary size and structure. This requires a search for the optimal pre-processor of arbitrary size.

Hypothesis 4.4 *Search techniques that employ a population of solutions are most appropriate for a search over pre-processors of arbitrary size, structure and constituent functions, and are able to focus on models with the right level of complexity.*

If this hypothesis is true, then for reasonably simple underlying models, the search space is effectively reduced to a manageable size.

Many questions arise from the points made so far; the questions addressed in this thesis are:

Question 4.1 *Is a search over pre-processors of arbitrary size, structure and constituent functions feasible for real problems?*

Question 4.2 *Is the generation of an economically-represented pre-processor useful for knowledge discovery?*

A section is dedicated to each of these hypotheses and questions below, but first the scope of this thesis is defined.

4.5 Scope of this Thesis

The concept of automatic pre-processing of data for classification opens a multitude of avenues for research. This thesis constitutes a first step, and does not attempt to address all of the issues at hand. The primary investigation of this work is the use of an evolutionary optimisation technique to search for a variable-sized network of interconnected non-linear, even discontinuous functions for pre-processing of data.

Pattern recognition has applications in every area of life, and only a sub-set of these problems can be addressed. The specific area of supervised classification was chosen for examination because of the availability of public domain data sets, and the presence of ground-truth information for evaluation of results. The wide variety of problems that can be examined in this genre facilitate general results, whereas focusing on face recognition or speech recognition would have limited the scope of the work.

The criticism may be leveled at this work that the synthesis algorithm used only works with low-dimensional data, so pre-processing is not needed anyway. The experiments had to be designed either for high-dimensional data or for low-dimensional data. There are several reasons why high-dimensional data were not used:

- Use of low-dimensional data allows for comparison with other classification methods.

- Manipulation of high-dimensional data requires exploitation of *regularity*. This issue has already been addressed in work on convolution networks, automatically-defined functions (Koza, 1994*b*) and the work of Teller (see Section 4.12).
- The use of low-dimensional data isolates the issues of interest, non-linear constituent functions and arbitrary structure, from the issues of regular structure and sparseness of the data.
- Work has already been performed by another researcher on the use of GP for automatic classification of images and sound waveforms. Use of high-dimensional data with EPrep would have resulted in work very similar to that of Teller (Teller and Veloso, 1996).

Given that mechanisms exist for exploiting regularity in high-dimensional data, the work done here can be considered as a search for what pre-processing should be applied to the data with regularity. This is literally the case in the **satimage** and **segment** problems to be encountered later (see Appendix C), which are image-segment classification problems.

The feature extraction methods considered in this thesis are functional compositions of constituent functions and input variables, and are therefore memoryless. Models that rely on previous inputs to the pre-processor, such as hidden Markov models and recurrent neural networks, are not considered. These models are useful for high-dimensional data, such as images and time-series, that are explicitly correlated. Most of the data sets examined in this thesis do not have explicit correlations between the input measurements. Hence there is no particular benefit to using pre-processor models with feed-back in the experiments of this thesis.

The method used to investigate Questions 4.1 and 4.2 is empirical rather than theoretical, the reason being that the questions are about the situation in the real-world rather than on paper. The traditional machine learning and artificial intelligence approach to evaluation of classification methods is empirical because the data and algorithms are usually too complex for a realistic formal treatment (Flexer, 1996). While experiments on synthetic data can be informative, and indeed are performed in the work of this thesis, experiments using real data are required to ascertain the practical utility of a system. Also, the use of empirical evaluation is similar to the prototyping methodology. We can quickly build a system to "see if it works", learn more about the problem and ascertain whether further investigation is worthwhile. If so, then we can return with a more formal treatment, smoothing the rough edges of decisions previously made via heuristics.

4.6 Hypothesis 4.1: Benefit of Appropriate Constituent Functions

This section addresses the validity of Hypothesis 4.1: can the use of appropriate constituent functions result in a more efficient pre-processor than existing methods of universal function approximation? The method will be to demonstrate situations in which this hypothesis is true. The relevance of this postulate in real-world situations is not addressed here. First, let us define what is meant by "appropriate constituent functions".

It is a commonly known fact that any periodic, piece-wise continuous function can be represented as the weighted sum of sine and cosine functions of different frequencies; this is the Fourier Series representation of the function. The basis functions used are sine and cosine functions, which are very important in engineering because they are the eigen-functions for linear time-invariant systems. The consequence is that, given sine and cosine functions, we can construct any function interesting to humans, and therefore any pre-processor. The problem is in the *efficiency* of the implementation, because these basis functions can be inappropriate in some situations. For example, a square wave function can be exactly represented a Fourier series, but an infinite number of basis functions is required. Obviously this is not practically realisable, whereas a finite set of threshold logic units could be used to represent the square wave over a finite interval.

The Universal Approximation Theorem (Theorem 2.1) for the MLP states a similar result, that any continuous function can be represented as a composition of addition, multiplication and the activation functions at the hidden nodes. Addition, multiplication and the activation functions can therefore be considered as the constituent functions for this pre-processing method. In a similar manner to the Fourier Series representation, the MLP can approximate any function *in theory*, but the practical implementation may be prohibitively large. To illustrate, consider the 2-class classification problem shown in Figure 4.1. There are 900 2-dimensional vectors, each belonging to one of two classes separated by the sinusoidal decision boundary $0.4\sin(2\pi f x_1)$, with $f = 15$ Hz and $x_1 \in [-1, 1]$. The data are divided into 200 training, 100 validation and 600 test samples.

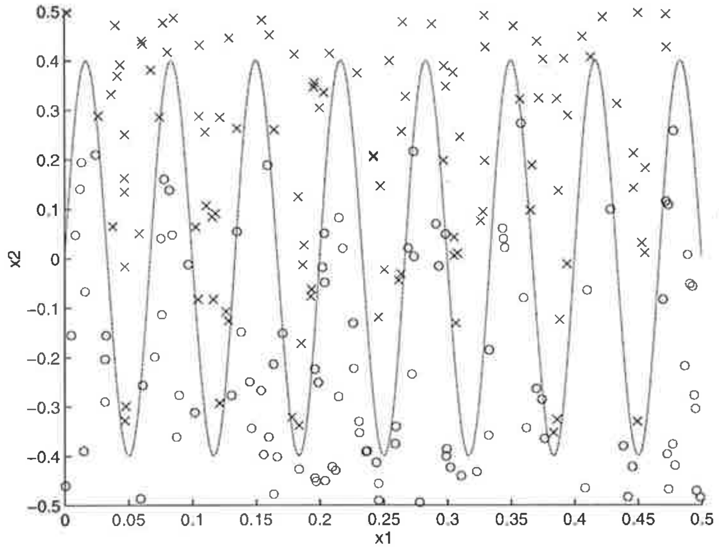


Figure 4.1: Synthetic sinusoidal boundary problem; the data set on the domain $[0, 0.5]$ (○: class 0, ×: class 1).

The MLP was used to classify this data. The experimental configuration is described fully in Section 6.3.3.1; in brief, the RProp algorithm (Riedmiller and Braun, 1993) was used to update the weights, and early stopping was used to halt training. To select the MLP architecture, 36 preliminary runs were performed: 6 single-layer and 6 double-layer architectures with up to 32 hidden nodes were searched, with 3 runs per architecture. Only connections between consecutive layers were used, and the target vectors were restricted to the $[0.1, 0.9]$ interval. The final results were obtained from 10 runs using the best architecture.

For this data set the MLP obtained a best test set misclassification error of 25.96%, and was clearly unable to approximate the complicated decision boundary with the limited number of hidden nodes used. If sinusoidal activation functions were used in the MLP, classification may have been more successful.

Numerous other scenarios can be engineered to make classification difficult for the MLP, such as the **spirals** problem described in Appendix C. The solution is to use *constituent functions that are appropriate for the problem*, but this generally prohibits successful training of the network. First of all, the activations must be everywhere differentiable, and second, the use of arbitrary continuous functions such as the sine function may introduce too many local optima to facilitate successful training.

Further proof for the validity of Hypothesis 4.1 comes from Kolmogorov's theorem. In answer to the 13th Hilbert problem, Kolmogorov proved that every continuous function of several variables can be written as the super-position of a fixed number of functions of one variable (Bishop, 1995, pg. 137). This theorem has had limited relevance for neural networks because the uni-variate functions are generally not smooth and their choice relies on the function being approximated (Bishop, 1995). The fact that a fixed number of functions is required for the Kolmogorov decomposition means that the mapping is practically realisable, except in cases where the univariate functions themselves are extremely complicated. The theorem therefore endorses the proposed framework of the use of arbitrary smooth or non-smooth constituent functions for the efficient realisation of a pre-processor.

4.7 Hypothesis 4.2: Benefit of Appropriate Structure

Now we turn to Hypothesis 4.2, that a pre-processor having the appropriate structure can be preferable to a traditional uniform structure. Let us first explore what is meant by "structure". The structure of a pre-processor defines the functional composition of the outputs, whereas the constituent functions define what goes into the composed functions. Any composition of functions can be represented as a feed-forward network of interconnected nodes, the nodes representing the functions and inputs, and the interconnections denoting the relationship of a function argument. In Chapter 2 it was seen that the traditional MLP consists of a number of layers of neurons, each of which is connected to every neuron in the preceding layer. For a given number of layers and neurons, however, there is no reason to believe that this interconnection strategy is preferable to any other: we could connect each neuron to every neuron in each previous layer, or only connect to a sub-set of the neurons in the preceding layer. One must also select the number of hidden layers and the number of neurons in each layer. The structure of the MLP is the combination of number of hidden layers, the number of nodes in each layer, and the matrix of interconnections between the neurons.

The concept of structure is not relegated only to neural networks: there are numerous other methods of data transformation which have some kind of structure. Decision trees and genetic programs are examples where the solutions have structure that is automatically obtained. For traditional neural networks, however, the structure must be selected by hand before training. The selection of the correct structure is very important, because it directly determines the complexity of the model. If the complexity of the structure is not concomitant with the data, then under- or over-fitting is likely to occur.

Note that the Universal Approximation Theorem states that the same structure, namely a single hidden layer, is sufficient for all continuous mappings, although the optimal size of the structure can vary. But once again the existence theorem is not always compatible with practice. A double-hidden-layer network may achieve the same mapping as a single-hidden-layer network but with fewer nodes. Similarly, a more economical representation may be achieved by reducing or increasing the number of interconnections between nodes; examples are found in (Fiesler, 1993). We have already seen an example of appropriate structure in the convolutional network, which, by reducing the number of inter-connections, achieves what the traditional MLP cannot within the bounds of current-day computational resources. The sheer weight of research into automatic methods for the choice of network structure is sufficient evidence that the optimal network structure is considered to be important (Branke, 1995; Bartlett, 1994; Miller *et al.*, 1989; Balakrishnan and Honavar, 1995*a*; Balakrishnan and Honavar, 1995*b*; Chen *et al.*, 1994; Zhang *et al.*, 1995; Bornholdt and Graudenz, 1992).

4.8 Hypothesis 4.3: The Generalised Pre-Processor

The case for Hypotheses 4.1 and 4.2 have been argued. On the basis of these hypotheses, Hypothesis 4.3 is that a pre-processor with flexible structure and composition has particular benefits in some situations. The generalised pre-processor is proposed as a framework for constructing pre-processors with arbitrary structure and using arbitrary functions. The GPP is simply a functional composition consisting of constituent functions selected by the user, and the input measurements as input variables. The GPP can be represented as a feed-forward network similar to the MLP. The differences from the MLP are that the architecture can have any number of layers, nodes and inter-connections, the inputs are not weighted, and the functions acting on the inputs to the nodes can take any non-linear form, and even be discontinuous. The GPP is a generalisation of the MLP, and so in theory it can perform any continuous mapping. The important feature of the GPP is that it is realisable with finite resources. This feature is made possible if the appropriate constituent functions for the problem are available.

Under the GPP framework, a criterion for an optimal pre-processor can be developed. Since the pre-processor is designed for use with a classifier, a candidate criterion is minimisation of the estimated misclassification rate of the classifier on the pre-processed data. There is an infinite number of pre-processors that yield the same estimate of misclassification rate for a given problem. The simplest pre-processor is chosen as the optimal one, because it is the most practically realisable. The concept of "simplest" here is some compromise between the complexity of the activation functions and the complexity of the pre-processor's structure.

By comparison of Kolmogorov's theorem with the Universal Approximation theorem for single-hidden-layer sigmoidal networks, it may appear that there is a duality between the constituent functions and the structure of the pre-processor. That is, we can fix the constituent functions and optimise the structure, as with the MLP, or we can fix the architecture and modify the constituent functions, as stated in Kolmogorov's theorem. This duality is false, however, because Kolgomorov's constituent functions can be arbitrarily complex and may therefore have a rich compositional structure of their own. Rather than a duality there is a trade-off between the complexity of structure and constituent functions: if we fix the structure to be small, we may need very complicated constituent functions, and if we use simple constituent functions we may need a complex structure. For instance, we could choose to build a pre-processor from addition functions, in which case multiplication would have to be represented by multiple additions: the simpler addition functions are accompanied by an increase in structural size. The practically desirable compromise lies somewhere in between, and is dependent on our level of representation of the functions. This also affects our concept of the "simplest" pre-processor mentioned earlier. The choice of constituent functions to use in constructing the pre-processor could be affected by the following considerations:

- Choose functions which have been known in the past to be useful for pre-processing; *eg:* Fourier transform, Gabor functions.
- Use functions which are in some way complementary, having different types of non-linearity.
- The functions should be meaningful to humans so that the results can be understood.

4.9 Hypothesis 4.4: Advantage of Population-Based Search

There is an infinite number of ways to represent a single functional mapping. The whole point of the generalised pre-processor is that it must be realisable using finite resources. The problem is, there is no known method for the synthesis of the optimal generalised pre-processor;

in most cases, the induction of minimal structures is NP-Hard (John *et al.*, 1994). A search technique is required which allows the combination of arbitrary constituent functions and the selection of an appropriate structure. The possibility of non-differentiable constituent functions and variable-sized structure precludes the use of many traditional search techniques. Only heuristic search can be used to manipulate totally general pre-processors, since gradient information is not explicitly available. Although the infinity of functions and structures cannot be fully considered when searching for a pre-processor, the quest can be successful because we are only interested in realisable pre-processors, which constrains the search.

The question arises as to which heuristic search technique to use. Let us define each pre-processor a_i by the pair $a_i = (\lambda_i, \theta_i)$, where $\lambda_i \in \Lambda$ is the structure of the model, its size and interconnectivity, and $\theta_i \in \Theta_{\lambda_i}$ is the set of parameters, inputs and constituent functions of the model, referred to here as *content*. For a fixed structure λ_i there is a search space over different contents, the size of which increases with the size of λ_i. Since the appropriate structure for the pre-processor is not known beforehand, structures of many different sizes need to be considered initially. The danger is that, since there are generally many more pre-processors of larger size that perform a particular mapping than of smaller size, the search technique may become focused on larger structures which increases the size of the content search space, and makes discovery of a suitable pre-processor less feasible.

Optimisation algorithms that use a population of solutions to search in parallel, such as genetic algorithms, are referred to as *population-based* methods. The more traditional optimisation methods that employ only a single solution to conduct the search are termed **single-point** algorithms. Hypothesis 4.4 postulates that population-based search techniques are more appropriate than single-point techniques for a search over generalised pre-processors because they are able to focus on models with the right level of complexity. If so, then for simple underlying models, the content space is effectively reduced to a reasonable size. The hypothesis is demonstrated here with some general analysis using the following notation and concepts.

There is a fitness function measuring the accuracy of the model, $f : \Lambda \times \Theta \to \Re$, with higher values of f being preferable and $f(a) \geq 0 \; \forall \; a \in \mathcal{A}$. Some measure of the size of individual i is denoted as s_i, and may be taken as the number of nodes in the pre-processor. We construct the situation in which the optimal solution with fitness f^* has a relatively small size s^*. The population-based algorithm begins with a population of M solutions with sizes $0 < s_1(0) < s_2(0) < \ldots < s_M(0)$, and s^* is one of the lower values of size. The algorithm is run for G generations, with the size of individual i at generation g being $s_i(g)$ and the population size M remaining constant. To ensure that the same number of points is visited by both methods, the single-point algorithm is run M times, and run i starts from individual i of the initial population of the population-based method so that a range of sizes is considered. Each run of the single-point method lasts for G iterations, and $s_i(g)$ is the size of the current solution at iteration g of run i.

The population-based method is considered first. The total size of the initial population is:

$$\bar{s}(0) = \sum_{i=1}^{M} s_i(0)$$

Let us assume that simple roulette-wheel selection is used in generating the next population. Parents are selected in direct proportion to their fitness and offspring are formed through the application of genetic operators:

$$a_i(g+1) = T(a_i(g))$$

All heuristic search techniques use a neighbourhood function to generate new search points in the vicinity of the current best solution; in the case of evolutionary methods, the genetic operators produce the neighbouring points, or offspring. We would expect pre-processors in the neighbourhood of a point to have a similar size and similar constituent functions and parameters to that point. Since the operators are applied stochastically, the neighbourhood function T is a stochastic process. Let us assume that T is unbiased with respect to size, so that $E_T[s_i(g+1)] = s_i(g)$, where $E_T[\cdot]$ is the expectation with respect to the stochastic function T. The expected total size of the population in the next generation is:

$$\begin{aligned} \bar{s}(1) &= \sum_{i=1}^{M} \frac{f_i(0)}{\bar{f}(0)} E_T[T(s_i(0))] \\ &= \sum_{i=1}^{M} \frac{f_i(0)}{\bar{f}(0)} s_i(0) \end{aligned} \tag{4.1}$$

where we have used $T(s_i(0))$ to denote the size of the offspring resulting from the genetic operator on individual i, and $\bar{f}(0)$ to denote the mean fitness in generation 0. Let us consider the sizes in the population to be a vector $\mathbf{s}(0)$, and similarly the corresponding fitnesses to be a vector $\mathbf{f}(0)$:

$$\begin{aligned} \mathbf{s}(0) &= [s_1(0), s_2(0), \ldots, s_M(0)] \\ \mathbf{f}(0) &= [f_1(0), f_2(0), \ldots, f_M(0)] \end{aligned}$$

Now we can write Equation(4.1) as a dot-product:

$$\begin{aligned} \bar{s}(1) &= \frac{\mathbf{f}(0) \cdot \mathbf{s}(0)}{\bar{f}(0)} \\ &= \frac{M}{\sum_{i=1}^{M} f_i(0)} . \cos(\theta(0)) |\mathbf{f}(0)|\ |\mathbf{s}(0)| \\ &= M \cos(\theta(0)) \frac{\sqrt{\sum_{i=1}^{M} f_i(0)^2}}{\sum_{i=1}^{M} f_i(0)} \frac{\sqrt{\sum_{i=1}^{M} s_i(0)^2}}{\sum_{i=1}^{M} s_i(0)} . \bar{s}(0) \\ &= M \cos(\theta(0)) F(0) S(0) \bar{s}(0) \end{aligned} \tag{4.2}$$

where $\theta(0)$ is the angle between the size and fitness vectors, $F(0)$ is the ratio of the length of the fitness vector to the sum of its elements, and similarly $S(0)$ is the ratio of the length of the size vector to the sum of its elements. We have the result that the total size of the individuals in generation 1 is some factor times the total size in the previous generation; this relationship generalises for all generations. Under what conditions does the total size decrease? We must have:

$$\cos(\theta(0)) F(0) S(0) < \frac{1}{M} \tag{4.3}$$

A sufficient condition for a geometric decrease in GPP sizes is:

$$M \cos(\theta(0)) < 1$$

We know from the triangle inequality that $0 \leq F(0) < 1$ and $0 < S(0) < 1$. The fitness and size values cannot be negative, therefore $0 \leq \cos(\theta(0)) \leq 1$. An example of a population of size 2 for which $\cos(\theta)$ is relatively small is shown in Figure 4.2. As can be seen from the figure, the angular separation between the vectors can be large if there is a good dynamic range of fitness and size values, and if there is an inverse proportionality between fitness and size. Although it has already been stated that there are usually more cases of near-optimum fitness for larger sizes, one would generally expect the fitness function to be more volatile for larger-sized structures. In such a case, the average fitness of smaller individuals will be higher than that of bigger individuals, and the conditions for a small $\cos(\theta)$ are viable.

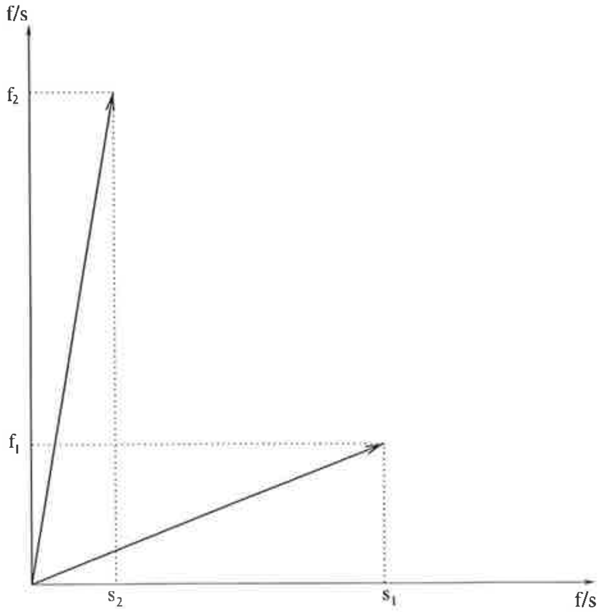


Figure 4.2: An example of size and fitness vectors for a population of size 2.

Equation(4.2) generalises to generation g:

$$\bar{s}(g+1) = K(g)\ \bar{s}(g)$$

where $K(g) = M\cos(\theta(g))F(g)S(g)$. If $K(g) < 1$ on average then there will be a geometric decrease in the size of the population as generations progress. As $s(g)$ approaches $M.s^*$, the mean size of individuals in the population will decrease. The sizes of the individuals cannot be negative, so the variance will also decrease. Therefore the population will become more homogeneous in size, and the size vector will approach a scaled version of the unit vector $[1, 1, \ldots, 1]$. Hence $\cos(\theta(g))$ will increase, and the average size of individuals in the population will stop decreasing.

To illustrate, simulation results showing the relationship between average individual sizes from one generation to the next for three different optimal solution sizes are shown in Figure 4.4. Using $M = 1000$, the sizes in the initial population were randomly generated 2000 times. For each of these populations, the total population size in the next generation was calculated using the piecewise-linear fitness function:

$$f(s) = \begin{cases} 100.\frac{s}{s^*} & \text{if } s \leq s^* \\ 100.\frac{s_{max}-s}{s_{max}-s^*} & \text{otherwise} \end{cases}$$

where s_{max} is the size of the largest solution in the population. This fitness function is shown in Figure 4.3 for $s^* = 30$. Note that the function quantifies the average fitness of individuals at that size, and larger values of fitness are better. By comparing the plotted points in Figure 4.4 with the line $\bar{s}(0) = \bar{s}(1)$, it can be seen that the average size of individuals decreases when greater than s^*, increases when less than s^* and stays the same when equal to s^*. Hence the change in solution size is always such that the sizes iteratively converge upon s^*.

The reason a geometric convergence upon the optimal solution size is possible with population-based methods is that solutions with inappropriate size, and therefore relatively low fitness, are discarded from the population on subsequent generations through natural selection. Regions of the search space that are found from the initial population to hold the

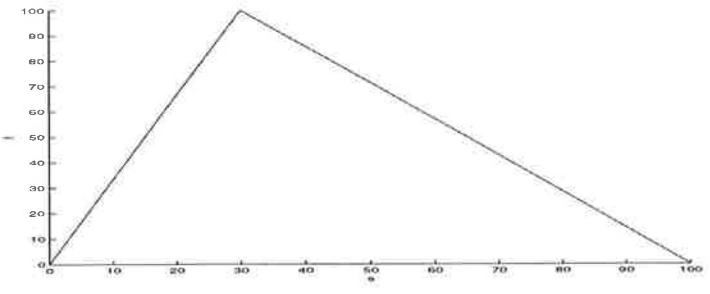
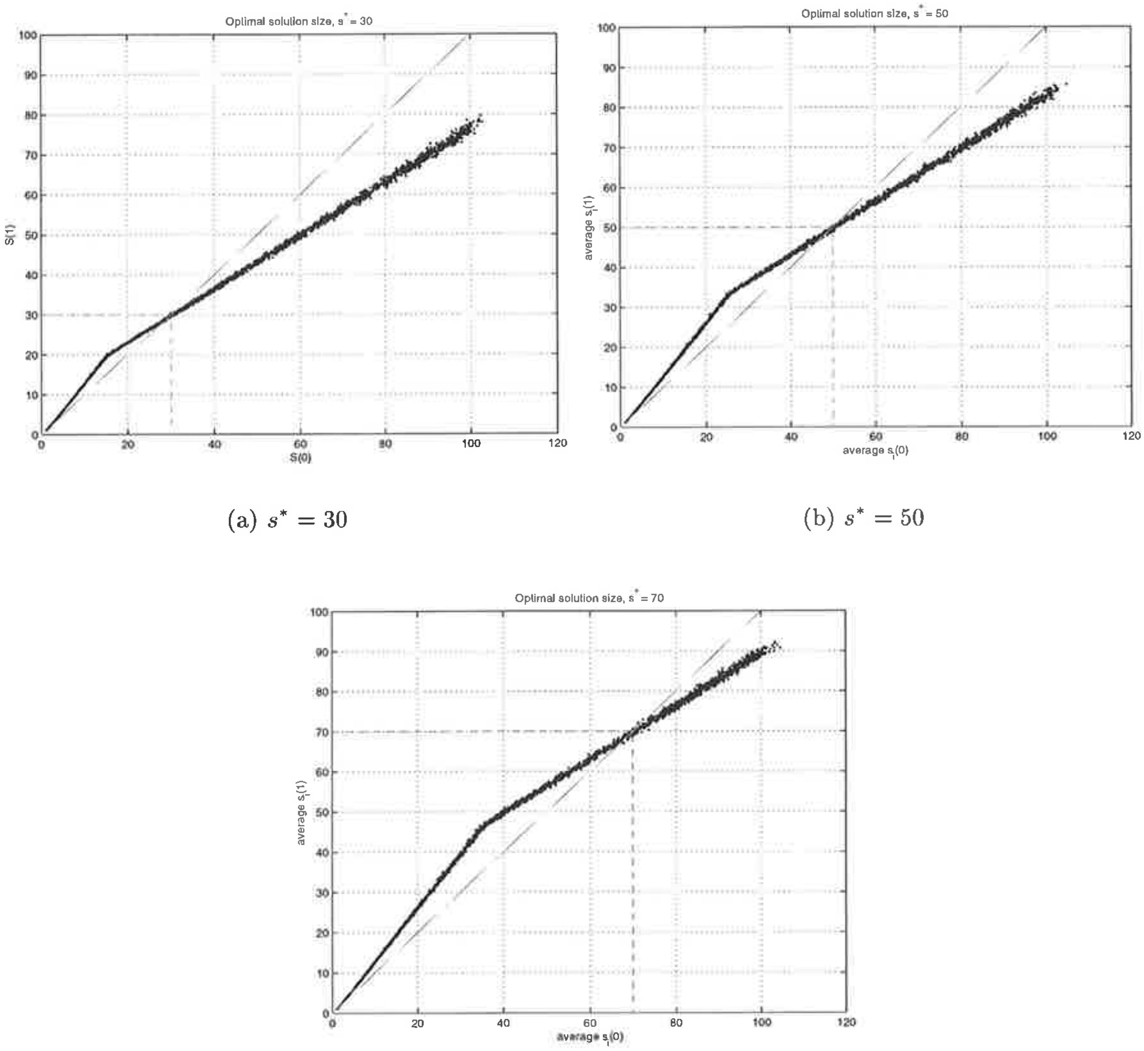


Figure 4.3: The average fitness of individuals with a given size plotted for $s^* = 30$.



(a) $s^* = 30$

(b) $s^* = 50$

(c) $s^* = 70$

Figure 4.4: Simulation results showing the change in size from one generation to the next for different optimal solution sizes.

most promise can have a proportionate amount of resources invested in them in subsequent generations.

Compare the population-based approach with the single-point search. The neighbourhood function used to generate new points is $T(.)$, the same as that used to generate offspring in the population-based method. Therefore on average, half of the neighbouring points will have a smaller size than the current point. Assume the best case for the single-point algorithm: that each neighbouring point under examination with smaller size than the current best point has higher fitness and becomes the new best solution, and that neighbouring points with larger size have lower fitness. Therefore, beginning with $s_i(0) > s^*$, the size of the best solution after g iterations will be:

$$s_i(g) = s_i(0) - \sum_{j=1}^{g/2} \delta_i(j)$$

where $\delta_i(j)$ is the size change at iteration j of run i. The total size of all solutions at iteration g over the M runs is:

$$\begin{aligned}\bar{s}(g) &= \sum_{i=1}^{M} \left[s_i(0) - \sum_{j=1}^{g/2} \delta_i(j) \right] \\ &= \bar{s}(0) - \sum_{i=1}^{M} \sum_{j=1}^{g/2} \delta_i(j)\end{aligned}$$

which, assuming the $\delta_i(j)$ are roughly the same on average, is an arithmetic decrease in size. This is compared with the geometric decrease in size associated with population-based search. Therefore the single-point search wastes resources on the larger initial pre-processors, for which the content space is also large and the likelihood of finding a near-global optimum is smaller. The reason single-point search suffers in this way is that the runs are independent, so that if the search begins in a relatively unfit region of the search space, it is stuck there for the rest of the run. In the population-based case, these unfit individuals are replaced with the offspring of the fitter individuals.

Note well that the argument here is the *possibility* that population-based methods can out-perform single-point methods by focusing on models of the appropriate size. Whether or not this will be the case in practice is dependent on the problem, and in particular whether Equation(4.3) is satisfiable.

The astute reader may have noticed that the hypothesis presented in this section appears to be in contradiction to the bloating phenomenon described in Section 3.5.6. Bloating occurs when the fitness and size vectors are correlated in the following way: solutions of various sizes can all share the same fitness value, and more importantly, there is a path between these solutions via the genetic operators. More specifically, via sub-tree crossover one can add to a genetic program a sub-tree that does the same job as the previous sub-tree, but which is much larger. Bloating is avoided in the experiments of this thesis by reducing the number of these constant-fitness paths from smaller to larger individuals via the genetic operators.

4.10 Question 4.1: Feasibility of Automatic Feature Extraction

The whole point of the general pre-processor is that it is practical rather than hypothetical. It would be contradictory to propose a framework that is of no practical use. Hence Question 4.1: is the search over generalised pre-processors feasible for real problems? By "feasible" we mean that the probability of finding an appropriately near-optimal solution is acceptable. It is not difficult to suspect that the search is infeasible when the size of the search space is considered. Suppose the pre-processor networks are represented as trees of minimum depth 1 and some maximum depth D. Each internal node of the tree can be one of F functions, and each leaf node can be one of V input variables or numerical constants. Let us also assume that all functions take two arguments so that the trees are binary. We can start to write the number of distinct trees for each depth:

$$\begin{aligned}
n(1) &= T \\
n(2) &= F.T^2 \\
n(3) &= F.n(2)^2 \\
n(4) &= F.n(3)^2 \\
&\vdots \\
n(D) &= F.n(D-1)^2 \\
&= F^{2^{(d-1)}-1}.T^{2^{(d-1)}} \\
&= \frac{(F.T)^{2^{(d-1)}}}{F}
\end{aligned}$$

Therefore the total number of distinct pre-processors of depth between 1 and D inclusive is:

$$\begin{aligned}
n_T(D) &= \sum_{d=1}^{D} n(d) \\
&= \frac{1}{F}\sum_{d=1}^{D}(F.T)^{2^{(d-1)}} \qquad (4.4)
\end{aligned}$$

which increases with depth as an *exponential of an exponential function.* To fix ideas, the number of possible trees is listed in Table 4.1 for various values of D using a modest $F = T = 5$. It is not unreasonable to expect pre-processors for real problems to have more than five functions or terminals, or a depth larger than 8, so how can one expect to search this space with any success?

Table 4.1: Number of possible binary trees with depth less than or equal to D and consisting of 5 functions and 5 terminals.

D	1	2	3	4	5	6	7	8
$n_T(D)$	5	125	78,125	3.052×10^{10}	4.657×10^{21}	1.084×10^{44}	5.878×10^{88}	1.727×10^{178}

The method chosen to answer this question is empirical experimentation. Much effort could have been expended by formally analysing a real-world problem, but the simplifications required to make the analysis tractable would cast suspicion on the validity of the results. Also, it is preferable to explore this question *for as many real-world problems as possible*, since results obtained for one problem may be totally different from those on other problems, but would nevertheless remain in the mind of the reader as a characterisation of the technique. The software required to perform the investigative simulations is called the *Evolutionary Pre-Processor* (*EPrep* for short). The total algorithm is very complicated and incorporates much knowledge gained through iterative investigation; it is described fully in Chapter 5. The question of whether EPrep can effectively search for pre-processors is discussed in Chapter 6, where comparative experiments are performed on real data using EPrep, neural networks, decision trees and a series of other statistical classification methods. The following subsections discuss some of the issues associated with EPrep.

4.10.1 Genetic Programming for Automatic Feature Extraction

To find an acceptable solution to an $\mathcal{NP}$-hard or $\mathcal{NP}$-complete search problem, special properties of the objective function must be exploited. There are two areas of exploitation that appeal to intuition. Firstly, the efficient convergence to the right-sized solutions offered by population-based algorithms, as stated by Hypothesis 4.4. Secondly, the intuitive semi-independent contribution of features to discrimination. It is a simple fact that a mapping from a d-dimensional space to a g-dimensional space is essentially the concatenation of g mappings from a d-dimensional space to a 1-dimensional range. In this respect, a pre-processor can be viewed as a *set of features*, each of which maps from the measurements to a single feature dimension. Each feature, when used on its own for classification, has its own ability to discriminate the data. Although not necessarily true, it is an intuitive step to assume that some features have greater discriminatory power than others, and that the addition of some feature to an existing set can improve the performance of the resulting set more so than a feature of lesser discriminatory power. This semi-independent contribution to fitness of a portion of the pre-processor is reminiscent of the building block hypothesis in genetic algorithms. Therefore since a population-based approach is already suggested, a recombinative method should also be employed to exploit the possible building blocks in the solutions.

The generalised pre-processor to be optimised is an arbitrarily-connected feed-forward network of different non-linear functions. Such a network can be represented as a forest of trees: one for each of the d-dimensional to 1-dimensional functions described previously. The whole field of *genetic programming*, described in Chapter 3, is devoted to the manipulation of tree structures containing arbitrary functions at the internal nodes. Hence GP was chosen as the general method to search for pre-processors. The final algorithm deviates significantly from the standard genetic program, but still manipulates the same variable-sized tree structures. GP uses recombinative search to exploit possible super-positioning effects with the features, and does not require explicit gradient information from the objective function, which is a necessary property to allow arbitrary constituent functions in the pre-processor.

4.10.2 Pre-Processor Representation

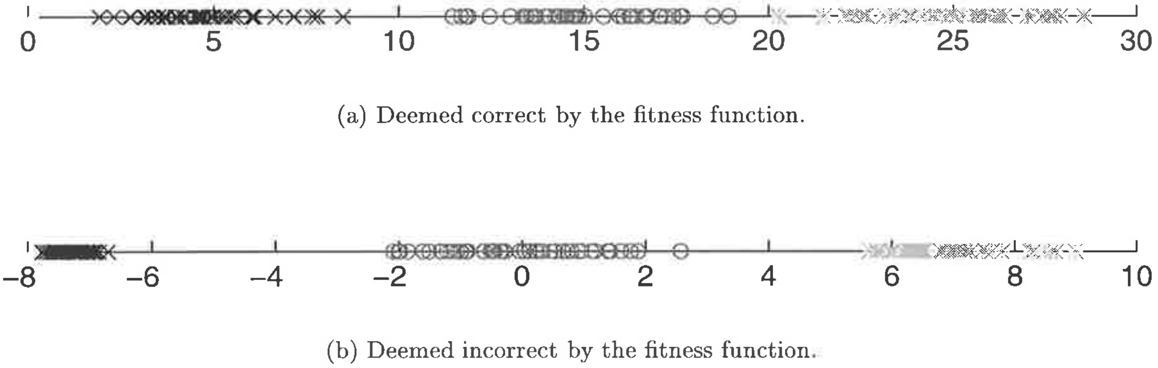
The functions of which the features are composed are selected to be complementary and to handle different types of data; they are described fully in Chapter 5. For reasons to be made known in the next section, the constituent functions are also commonly used by humans. There are arithmetic functions, such as addition, multiplication, absolute value and exponentiation, boolean functions such as AND, OR and NOT, and miscellaneous functions such as if-then-else and test for equality. The arguments to the functions can be the outputs of other functions, the input measurements for the problem, and randomly-generated constants. It would be ideal to throw in every function known to man and have the algorithm determine the best combination of these. In practice, GP is severely limited by the exponential growth in search space with the number of functions and terminals (see Equation(4.4)). Therefore the dimensionality of the data is severely restricted, and images or time-series cannot be used directly with EPrep.

4.10.3 The Role of the Classifier

It is very important that the pre-processor be constructed to complement the classifier. If the pre-processor were developed independently of the classifier, then the objective criterion used could be completely different from that used to train the classifier. For example, a classifier that assumes only one cluster per class must be preceded by a pre-processor which collects all the clusters from one class into a single cluster. The GP approach allows the pre-processors and classifier to be considered together by basing the fitness function on an empirical estimate of misclassification cost using the classifier on the pre-processed data.

It may be argued that no classifier is required at all: since the pre-processor performs a multi-dimensional mapping, why not have it approximate the mapping directly to the predicted class labels? Indeed, this approach has been taken previously using GP (Tackett, 1994). In that work, vehicle detection was framed as a two-class classification problem. Each individual in the population yielded a single feature, which was thresholded to obtain a "present/absent" decision. This idea was generalised to C classes by (Banzhaf *et al.*, 1996), in which the range of the output value was divided into C regions of equal size. The region to which the output fell corresponded to the predicted class of that sample.

The method of arbitrarily dividing the output range into C fixed intervals is referred to here as a *static decision rule*. The static decision rule approach is not used here because it is not as general as the adaptive decision rule realised by a trainable classifier. The local tuning performed by the classifier facilitates search by removing the reliance of the pre-processor fitness on the arbitrarily-placed decision region boundaries. Take the example in Figure 4.5(a) below: three classes, with means of 5, 15 and 25 along the single feature, fall into the regions 0-10, 11-20, 21-30. The fitness function designates that each sample falling in the 0-10 region comes from class 1, and so on, so that 100% correct classification is achieved. Now consider the data shown in Figure 4.5(b) which have been transformed by some pre-processor: although this solution is *semantically* correct in that it separates the data into distinct clusters, it is judged to be *syntactically* incorrect by the objective function because the samples do not fall in their arbitrarily-placed regions. A classifier with adaptable decision boundaries, on the other hand, introduces a shift and scale invariance that can recognise semantically correct solutions regardless of the irrelevant scale and shift factors. This provides the GP with more ways of representing desirable solutions, and therefore increases the likelihood that the GP will converge to a good solution.



(a) Deemed correct by the fitness function.

(b) Deemed incorrect by the fitness function.

Figure 4.5: An example of the static decision rule.

Another reason why a classifier is useful is that it may allow the pre-processor to be simpler in some situations. For example, if the classification problem has a parity-type structure, a pre-processor to perform the full task of discrimination may be quite complicated and difficult to design. If, however, a classifier is used which can discriminate multiple clusters per class, then the required pre-processing is simpler and easier to find.

4.11 Question 4.2: Automatic Feature Extraction and Knowledge Discovery

The field of *knowledge discovery*, or *data mining*, has received extraordinary popularity of late. The basic principle is that large databases can be automatically processed by computers to find succinct pieces of information which are "interesting" to people (Berry and Linoff, 1997). Example applications are fraud detection in banks, market segmentation for direct marketing, and medical diagnosis. The most classic example of its use is in the finding that nappies and beer were often bought simultaneously in a chain of supermarkets. The explanation is that fathers are performing this transaction. Despite the simplicity of this pattern, there are relatively few such real-world examples. Techniques from statistics and machine learning are widely used in data mining, such as clustering, decision trees, neural networks and genetic algorithms. The enormous databases must first be reduced using simple methods to a size which renders the application of these methods feasible. The learning algorithm is used to form some sort of model of the data through regression, classification or clustering. The model is then used for prediction, or scrutinised to reveal some underlying patterns in the data.

There are an infinity of ways to describe a finite set of data. Therefore one may wonder that the user could be at all certain that the model obtained is the true model responsible for generating the data. The principle of Occam's razor is generally employed in data mining: that among a set of models that accurately describe the data, the least complex of those models is the most likely to be responsible for its generation. Given that the optimal generalised pre-processor among those with adequate performance is the simplest according to some criterion, it is possible that the resulting pre-processor could be reverse-engineered to learn about the problem.

Question 4.2 asks whether EPrep would be useful for knowledge discovery. It is generally agreed upon that the MLP is not useful for revealing underlying relationships, because the model must be apprehensible by a human, and a set of real-valued weights reveals little. Decision trees, on the other hand, are useful for knowledge discovery because they generate a series of questions or tests on the data which are easily understood by a person. The problem with decision trees is that when the clusters are not axis-parallel or contain non-linear dependencies between input variables, large complicated trees result and interpretability breaks down.

EPrep has been designed to investigate Question 4.2 by taking the following design measures:

- constituent functions are used that are meaningful to humans;
- the representation of features is simple to understand;
- the boundary between features is clear because they are represented as separate trees; and
- in the context of equal classification accuracy, there is a bias towards simpler pre-processors.

There are two main difficulties with the interpretability of solutions generated by EPrep:

- the way in which the features work with the classifier to perform discrimination may not be clear; and
- the GP trees generally contain superfluous material which makes no contribution to the overall fitness of the individual.

The empirical investigations in Chapter 6 will serve to illuminate EPrep's utility for knowledge discovery. In particular, several synthetic problems will be used for which the true underlying model is known.

4.12 Previous Work using GP for Pattern Recognition

Most of the previous work on classification using GP has been performed for a specific application or domain. Classification of low-resolution optical characters has been achieved by co-evolving 2-D features and the algorithms for using them (Koza, 1994*b*; Andre, 1994). A similar approach was taken to the classification of protein segments in (Koza, 1994*b*). Tackett (Tackett, 1993) evolved expressions to combine pre-calculated statistics of infra-red images for the detection of military vehicles.

The *Compiling Genetic Programming System* has been used by (Francone *et al.*, 1996) on three classification problems to investigate generalisation capabilities as the training set size is reduced. GP has also been used to classify data with missing values (Backer, 1996), and to classify brain tumours from nuclear magnetic resonance biopsy spectra (Gray *et al.*, 1996).

Evolutionary computation has been used extensively for the design and training of neural networks; an overview of this field is found in (Branke, 1995). In particular, genetic programming has been applied to the evolution of neural networks for classification by Zhang and Mühlenbein, who combined GP with hill-climbing to fine-tune the weights (Zhang and Mühlenbein, 1995), and also by (Friedrich and Moraga, 1996), who used the *Cellular Encoding* representation of (Gruau, 1992).

Teller and Veloso developed the evolutionary *Parallel Architecture Discovery and Orchestration* (PADO) system for the classification of arbitrary signals. The idea is similar to that of EPrep: a system to automatically generate a program that can identify signals, so that the wheel does not need to be re-invented for every classification problem. PADO has been applied to sound and visual object recognition (Teller and Veloso, 1995*b*), and human face recognition (Teller and Veloso, 1995*a*). PADO evolves programs that recognise individual classes, and combines the outputs of these programs to give an overall decision. The PADO system is extremely complicated, so a full description cannot be given here. Since there are similarities between PADO and EPrep, the main differences are listed:

- EPrep is designed to work with medium-sized databases of small signals, such as social science databases, whereas PADO is designed for small sets of large signals, such as sounds and images.

- PADO uses loops and indexed memory to detect and exploit regularities in the data; this is how it can classify such high-dimensional signals.

- EPrep is designed to work with mixed data types, while PADO works with integers only.

- PADO evolves the whole classification program, while EPrep searches for the pre-processor only.

PADO is an exciting concept with motivations similar to the work of this thesis. However, it is the author's view that the data used to test PADO are so sparsely-distributed in the very high-dimensional space that the accuracy of misclassification cost estimates must be called into question. The work of this thesis tackles the issues of feature extraction at a lower problem scale so that more realisable results can be obtained.

4.13 Conclusion

The main ideas that have motivated the work of this thesis have been presented in this chapter. In brief, a general framework for pre-processing is required to further the field of pattern recognition. This requires a functional mapping that is not only universal, but is also realisable. It is the proposition of this thesis that feed-forward networks of arbitrary structure and composed of different non-linear constituent functions constitute such a framework. There is, however, no known algorithm to synthesise the optimal realisable pre-processor. The evolutionary pre-processor has been developed as a first attempt to search over the space of these pre-processors. Although there is an infinite number of realisable pre-processors with similar performance for a given problem, preference is given to less complex models according to Occam's principle.

The next chapter describes EPrep, the system used to search for an appropriate pre-processor. Chapter 6 details the experiments used to investigate the feasibility of a search over variable-sized pre-processors and their use in knowledge discovery. The conclusions are given in Chapter 7.

Chapter 5

The Evolutionary Pre-Processor

5.1 Introduction

One of the questions posed in the previous chapter is whether a search over variable-sized pre-processors with arbitrary constituent functions is feasible for real problems. The *evolutionary pre-processor* (EPrep) is a piece of software which has been designed and implemented to carry out the experiments necessary to answer this question. The purpose of this chapter is to describe the EPrep algorithm and its components in detail. The details of the software implementation are generally irrelevant to the algorithm itself, and are not included here. Some of the implementation details and software design choices are included in Appendix B for the curious reader.

5.2 Algorithmic Design Issues

Before examining the details of the algorithm, it is necessary to describe some of the overall design issues that are independent of the implementation: namely the representation of solutions, the population model and the objective function.

5.2.1 Solution Representation

Each individual in the population must represent a variable-sized arbitrarily-connected network of nodes with arbitrary transfer functions, and a classification algorithm. The classification algorithm is specified by a label indicating one of a fixed set of classifiers available; its use will be further described in Section 5.2.3. The remainder of this section focuses on the representation of the pre-processing networks. There is no unique representation for this task; in choosing a representation the following criteria must be considered:

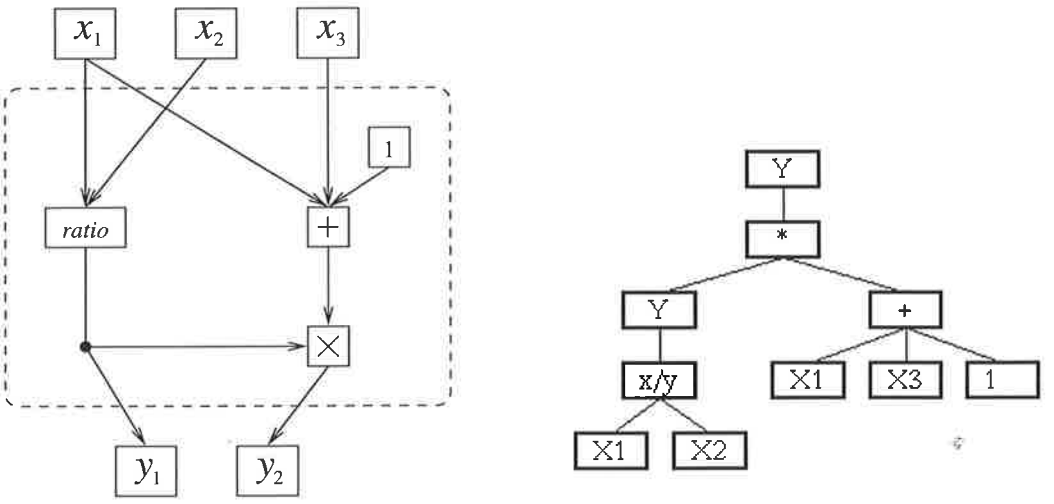
1. it should allow simple modifications by genetic operators that result in valid solutions;

2. the migration of highly-fit short-defining-length building blocks should be facilitated;

3. the solutions should be easy for a human to read and interpret.

Note that it is the combination of representation and operators that facilitates efficient search, so in principle any representation can achieve the same results by appropriate modification of the genetic operators. The operators can, however, become quite complicated which may result in programming or logical errors. For instance, suppose we represented each solution as a directed acyclic graph; it is not obvious how recombination would proceed, since the crossover fragments would generally have different numbers of input and output links which would be nonsensical in their new context.

The easier approach is to represent the network as a tree: this is the representation used in EPrep. It is instructive to consider the representation used in the previous version of EPrep and its associated flaws. An example is shown in Figure 5.1. The Pre-Processor in Figure 5.1(a) represents a feed-forward network of diverse functional nodes. The Pre-Processor operates on the input vector $\mathbf{x} = [x_1, x_2, x_3]$ and transforms it to the output vector $\mathbf{y} = [y_1, y_2]$. The corresponding expression tree is displayed in Figure 5.1(b). The internal nodes of the tree are functions which take the outputs of their child-nodes as arguments. The leaf nodes are terminals which act as inputs to the Pre-Processor. Each function operates on a vector of real-values and outputs a vector whose length depends on the function and its number of inputs. Thus all edges in the tree represent vectors of arbitrary length; this approach allows complicated expressions to be represented succinctly. Each function operates on the concatenation of the output vectors of its child-nodes. All terminals are scalar-valued. The output features are generated as follows. Execution flows up the tree from the terminals to the root node, and the features are gathered at the *output points* labeled "Y" on the diagram. The left branch takes the ratio of x_1 and x_2 and stores it as the first feature. The right branch calculates the sum of x_1, x_3 and 1. The two branches' results are multiplied to produce the second feature. The use of the output points in this representation allows individual features to be transferred through crossover.



(a) Example of a Pre-Processor.

(b) Tree representation of the Pre-Processor.

Figure 5.1: An example of EPrep's Pre-Processor representation.

There are several flawed aspects of this representation:

Weak Typing: any node can be an argument of any function regardless of the compatibility of their types. For example, a continuous input variable can be the argument of a boolean `AND` function. The weak typing is often exploited by the GP to generate portions of code that are unnecessarily complicated. For instance, consider the expression:

$$\texttt{X1} < \frac{1}{\texttt{X2}}$$

If `X1` is a boolean variable and `X2` is real-valued, the meaning of the expression depends on the numerical range of the real variable and on the representation of the boolean quantity. The expression would be better expressed as:

$$NOT(\texttt{X1})\ \ AND\ \ \left(\left|\frac{1}{\texttt{X2}}\right| > \frac{1}{2}\right)$$

Vector branches: the fact that all inputs and outputs of functions are vectors can make a solution virtually impossible to understand, since the dimension of each vector is not displayed and must be determined by fully tracing the execution of the expression below the relevant branch.

Varying arity: [1] each function can take any number of arguments (greater than zero); this requires default behaviours which impede the interpretation of solutions.

Output points: although output points provide a building-block structure for the solution representation, their hierarchical placement is un-intuitive and impedes understanding of the solutions. Output points can also produce highly correlated features, which suggests an unnecessary redundancy and therefore sub-optimal parsimony.

The representation used in EPrep version 3.0 seeks to overcome these problems and addresses the 3 criteria mentioned earlier. The *multi-tree* representation is an array of expression trees, as shown in Figure 5.2. This representation has been used previously: in (Langdon, 1996*a*), data structures were evolved by adapting the set of functions for an abstract data type. Each function was represented by a single tree. Similar work by (Bruce, 1996) used an array of trees representing object methods to generate abstract data type objects. The representation has been used in the dynamic modeling of industrial processes (Hiden *et al.*, 1997), in the evolution of teams of co-operative agents (Luke and Spector, 1996), in the evolution of multi-input-multi-output dynamical system models for a communications receiver problem (Jaske, 1997), and for the classification of magnetic resonance spectroscopy data (Gray, 1997). Angeline has also used the representation for his *multiple interacting program* system (Angeline, 1997*a*).

This representation has been referred to in the literature as a *multi-chromosome*, but is here called a *multi-tree* chromosome to avoid ambiguities with biological nomenclature.

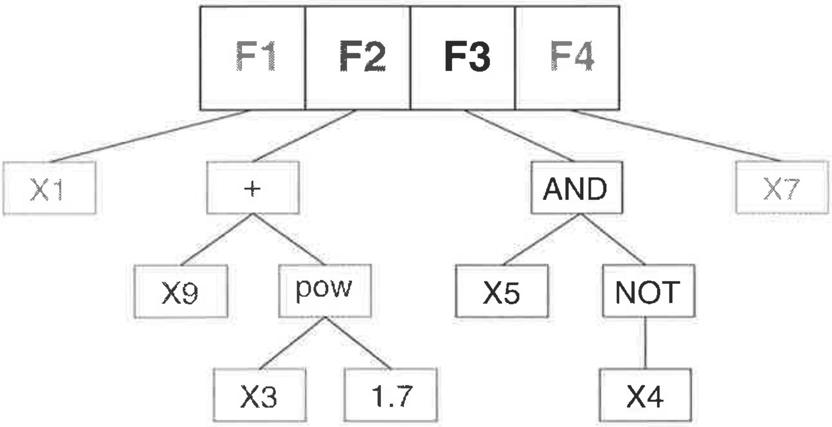


Figure 5.2: The multi-tree representation.

The motivation for using multiple trees is to permit the presence of multiple features that can be un-correlated and are clearly distinguishable for interpretation. The representation allows two scales of search: a search over combinations of features, and a search over the

[1] The *arity* of a function is the number of arguments it requires.

features themselves. As will be seen later in Section 5.8, recombination occurs at the feature level, while mutation occurs at the node level. Thus combinations of useful features can be manipulated as building blocks, while sub-trees within the features that do not contain obvious context-independent building blocks can be modified gradually through mutation.

The following constraints are applied to the multi-tree representation, and are maintained by the creation and genetic operators:

1. a feature cannot be a constant.
2. a genotype cannot contain two features that are identical.
3. a feature tree cannot exceed a user-defined maximum depth.
4. the length of the tree array cannot exceed d, the number of initial measurements for the problem. This ameliorates the curse of dimensionality and maintains generalisation. Also, this constraint avoids the bloating problem, because the number of non-contributory features which can be added to an individual is limited.

The feature expressions follow the traditional GP methodology of fixed-arity functions with scalar inputs and outputs. This does not preclude the construction of arbitrary functions, since multiple-argument functions can be equivalently constructed through compositions of fixed-arity functions. For example, $+(1\ 2\ 3\ 4) = +(1\ +(2\ +(3\ 4)))$. To further assist in the interpretability of solutions, the trees are strongly-typed: each terminal, function argument and function return value has a type (Boolean, Real or Enumerated) and a valid program must have all return value types matching with the corresponding argument types. The syntactic correctness of programs is maintained during initial random creation and the application of the genetic operators. Strong typing reduces the size of the search space as seen by the GP, because many isomorphic solutions are eliminated from the set of possibilities. See Section 3.5.3 for further details.

In EPrep, there are three node types used:

Real: any real number, including the integers.

Enumerated: an enumerated value which can take one of several values having no natural ordering, such as the colours $\{red, green, blue, yellow\}$.

Boolean: either *True* or *False*.

Although Enumerated constants in EPrep and in the data are represented as integers, they are not manipulated as normal integers (*eg:* with functions like addition and multiplication) since the values by definition have no natural ordering. If the order does have significance (*eg:* age in years, number of children) then the value should be treated as an integer, making it type Real. It is important to note that different enumerated variables are not considered to be of equal type, since they have different ranges of values. Therefore enumerated variables $X1$ and $X2$ cannot appear in the boolean expression $X1 = X2?$. As a result, the only expression an enumerated variable can appear in is $X = c?$, where c is a constant of type Enumerated within the range of values taken by X. An enumerated variable can appear on its own as a feature; *ie:* $F1 = X$.

The functions and terminals used in EPrep and their associated type specifications are listed in Table 5.1. It is up to the user to select an appropriate sub-set of these functions and terminals. All input variables for the problem are automatically included in the terminal set. Note that the return type for functions that can return one of several types (*eg:* If-then-else) is specified by the parent node in the program.

Table 5.1: Function and Terminal argument and return types in EPrep.

Function or Terminal	Return Type	Argument Types
Addition	Real	(Real, Real)
Subtraction	Real	(Real, Real)
Multiplication	Real	(Real, Real)
Power	Real	(Real, Real)
Ratio	Real	(Real, Real)
Absolute Value	Real	(Real)
Natural Logarithm	Real	(Real)
Exponentiation	Real	(Real)
Equal To	Boolean Boolean	(Real, Real) (Enumerated, Enumerated)
Less Than	Boolean	(Real, Real)
Logical AND	Boolean	(Boolean, Boolean)
Logical OR	Boolean	(Boolean, Boolean)
Logical XOR	Boolean	(Boolean, Boolean)
Logical NOT	Boolean	(Boolean)
If-Then-Else	Real Boolean	(Boolean, Real, Real) (Boolean, Boolean, Boolean)
Sine	Real	(Real)
Cosine	Real	(Real)
Real Input Variable	Real	–
Enumerated Input Variable	Enumerated	–
Boolean Input Variable	Boolean	–
Real Constant	Real	–
Ephemeral Random Constant	Real	–
Enumerated Constant	Enumerated	–

5.2.2 Population Model

Even though steady-state evolutionary models are becoming the norm in recent times, a generational population model is used to promote diversity, and to ensure that all individuals are evaluated using the same training sample order.

To maintain diversity, exact duplicate individuals are not placed in the population; therefore no reproduction operator is used, since it would place multiple copies of some individuals in the population. It is still possible for duplicates to occur as the result of genetic operations. Functional duplicates are too difficult to detect, and so the question of whether or not to exclude them will not be considered.

Some form of reproduction is still desirable to avoid the loss of highly-fit individuals through destructive crossover. In EPrep, the N_{rep} best individuals from the previous generation are copied straight into the new generation. This stops duplicates from being explicitly placed in the population, and leaves an effective $M - N_{rep}$ individuals for exploration.

5.2.3 Objective Function

The qualitative criterion for evolution is to generate a pre-processor that separates the data appropriately for the classifier used. While class-separability criteria are often used in unsupervised learning, decision trees and other methods, each criterion makes certain assumptions about the data which would not be applicable in all instances. Also, the pre-processor should co-operate with the classifier in its action on the data; this interaction may not be captured by the criterion used. These issues were avoided by basing the objective function directly on an empirical estimate of classification error rate.

The classifier used by each individual is one of a set of three algorithms chosen for their speed, simplicity, few parameters and independence of initial conditions and sample order:

- Minimum-Distance-to-Means Classifier (Section 2.6.2);
- Parallelepiped Classifier (Section 2.3.1); and
- Gaussian Maximum Likelihood Classifier (Section 2.6.3).

The classifiers must not be too powerful, otherwise they will do all the work of classification and there will be no pressure for EPrep to evolve useful features. The classifiers are to some extent complementary: the MDTM and PPD classifiers only allow one cluster per class, while the quadratic discriminant boundaries of the ML classifier allow multiple clusters for each class. Note that the MDTM and PPD classifiers are generally different, since the mean of the class samples is not necessarily the same as the centroid of the parallelepiped. Both are included because the PPD classifier is faster than the MDTM classifier *on average.* These classifiers have the advantage that only one training epoch is needed, making them fast enough to use in EPrep.

The following method is used to calculate the objective function (fitness) of an individual, which is to be minimised [2] by the evolutionary algorithm. Let the pre-processor function of individual i be $F_i(\mathbf{x})$. The training data are transformed to the feature space using the pre-processor of individual i:

$$\mathcal{P}_i = \{\mathbf{y}_j | \mathbf{y}_j = F_i(\mathbf{x}_j);\ j = 1, \ldots, n_{tr}\}$$

where n_{tr} is the number of training samples. Next the classifier associated with individual i is trained on $\mathcal{P}_i$, resulting in the classification rule $h_i(\mathbf{y})$. The fitness of individual i is the

[2] The fact that the fitness function is minimised in EPrep makes the meaning of the word "fitness" counter-intuitive. Fitness will nevertheless be used to refer to the objective function.

apparent error percentage on this transformed training set:

$$f_i = 100\% \times \sum_{j=1}^{n_{tr}} \frac{I(h_i(\mathbf{y}_j) \neq c_j)}{n_{tr}} \tag{5.1}$$

The computational cost of this objective function is linear in the number of training samples. Experience with previous versions of EPrep has shown that this complexity is a severe limitation on the size of problem EPrep can cope with. The *Rational Allocation of Trials* algorithm, described in Section 5.5, is used in EPrep 3.0 to relax this complexity somewhat. This algorithm evaluates each individual on the bare minimum number of training samples required to make the error estimate statistically accurate enough for the evolutionary algorithm. The true objective function obtained using this algorithm is more complicated than that described here, but for now the current description will suffice. It should be noted, though, that the first N_o samples are used in some situations to obtain a rough estimate of the apparent error, where N_o is a user-defined parameter.

The reader may be wondering at this stage how an individual that optimises this objective function can provide good generalisation performance. There are three mechanisms that assist in avoiding over-fitting on the training data:

Parsimony Preference: rather than parsimony pressure, since it is only a gentle bias towards less complex models. In comparisons involving two individuals having equal fitness, the less complex model is preferred. There are several ways to quantify complexity, such as the number of nodes in the individual n_n, the number of features n_f, and the number of distinct input variables n_v. Since the curse of dimensionality is such a problem, preference is given to individuals with fewer features. Parsimony in the number of measurement variables is also important from an interpretative point of view, so this is the next index for sorting. Finally the number of nodes is used to distinguish individuals. Overall the decision process is to select individual A in preference to B if:

$$\begin{array}{ll} & f_A < f_B \\ \text{or} & f_A = f_B \text{ and } n_f^A < n_f^B \\ \text{or} & f_A = f_B \text{ and } n_f^A = n_f^B \text{ and } n_v^A < n_v^B \\ \text{or} & f_A = f_B \text{ and } n_f^A = n_f^B \text{ and } n_v^A = n_v^B \text{ and } n_n^A < n_n^B \end{array}$$

Time-varying Objective Function: because a sub-set of the training samples is used to calculate classification error, the order of the samples is changed each generation according to the algorithm described in Section 5.6. Therefore an individual can only stay in the upper echelons of the population over a number of generations by performing well on a series of partially-different training sets. This characteristic requires good generalisation performance.

Early Stopping: for many problems, the best individuals in the population would eventually learn to classify the whole training set almost perfectly to the detriment of generalisation. We would never know at what point the solutions lost their ability to generalise. A validation set is used to detect this point in the optimisation, as described in Section 5.9. The solution from the generation at which the best-of-generation individual yielded the minimum validation set error is kept as the best-of-run individual.

5.3 The EPrep Algorithm

An execution flow diagram of the entire EPrep 3.0 algorithm[3] is shown in Figure 5.3. Each rectangular block represents some process which may consist of several sub-routines. Each diamond-shaped block represents a test as part of an if-then-else statement or as the condition of a loop. Execution leaves the conditional boxes from the left or right when the condition is true, and from the bottom when the condition is false. The arrows on connecting lines show the direction of execution. The program begins with the block labelled "EPrep" at the top, and ends with the block "end EPrep" below it. Most blocks in the diagram contain references to the sections in which a description and motivation for their use can be found.

An overall description of the algorithm is given here. The remaining sections describe the pieces of the algorithm *in order of their logical dependence*, and not the order in which they are encountered in the diagram.

An outer loop allows the algorithm to be repeated R times, and the best overall results are recorded. Each run begins with the initialisation of data structures, the most significant being the random creation of the initial population. Then the generational loop is entered until one of the termination criteria is reached. First, the enumerated and real-valued constants in each individual are locally optimised. Then the Rational Allocation of Trials (RAT) algorithm is applied to calculate the fitness of individuals and select individuals for the mating pool. With the fitness of each individual calculated, the population can now be sorted with lower values of the objective function being preferential and ties broken as described in Section 5.2.3. The best individual is noted as the best-of-generation (BOG) individual.

Now that fitness evaluation is over, the next generation can be formed. No more fitness calculations are required for this generation, so the training sample order is modified. Filling of the next generation's population begins with the transfer of the top N_{rep} individuals. The rest of the individuals are obtained by taking the next parent individual(s) from the mating pool and applying a randomly-selected genetic operator to obtain the offspring. Note that if the offspring is the same as the parent, no modification has occurred and the process of selecting and applying an operator is repeated to ensure that duplicates are not explicitly placed in the population. It is important that the operator selection be included in this loop because some operators cannot bring about a change in some individuals, and the repeated application of such an operator would cause an infinite loop. With the new population full, the operator probabilities contained in the individuals are perturbed randomly and the inversion operator is applied. The validation set error of the BOG individual is tested against that of the best-of-run individual (BOR) to see if this generation has yielded the best performance so far. This point marks the passing of a generation, and the next generation is entered.

At the end of a run, the BOR individual is first cleaned to remove superfluous features. It is then compared with the best-ever individual according to validation set error and size to check if it has become the best-ever individual.

5.4 Initial Population

The initial population is generated randomly and becomes the starting point for the optimisation process. It is therefore crucial that the population be initialised with care, since biases introduced in the initial population will inevitably bias the search algorithm. To obtain a wide range of shapes and sizes in the new trees, the *ramped half-and-half* generation method

[3] This is a list of steps performed by the algorithm, and does not include the steps that would have to be performed by the user to apply the algorithm to his or her data (*ie:* initial selection of functions and terminals, parameters, etc.).

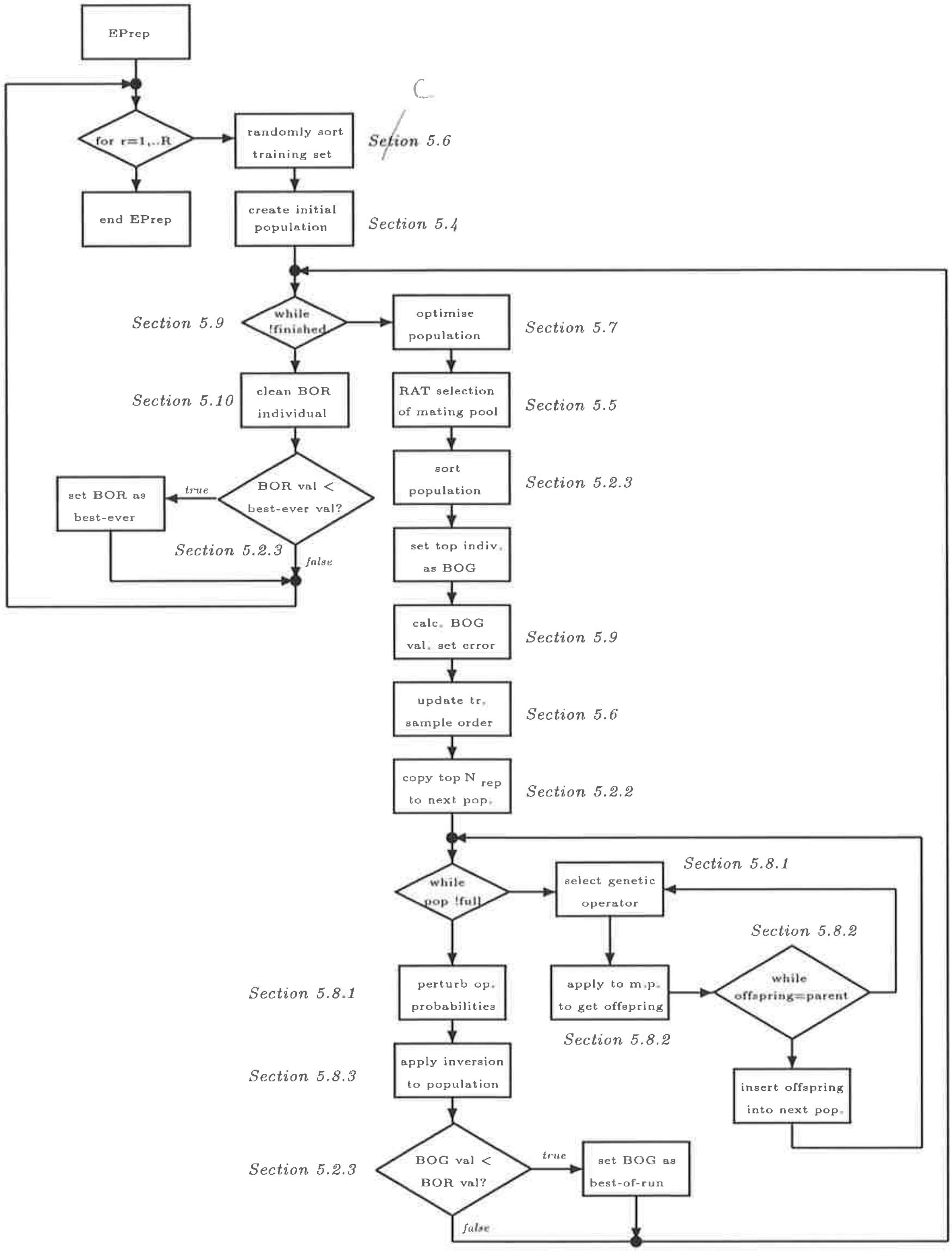


Figure 5.3: Execution Flow Diagram of the evolutionary pre-processor Algorithm, version 3.0.

is used (Koza, 1992*a*). This method generates equal numbers of solutions with depths ramping from 2 up to a user-specified maximum initial depth. Half of the individuals are created using the *full* method, and half using the *grow* method. The full method ensures that all terminals occur at the maximum depth of the tree (*ie:* all nodes at less than the maximum depth are functions), and the grow method allows functions or terminals at any point in the tree. The algorithm is described more fully in Section 3.5.1 of Chapter 3.

There may be cases in which the original input measurements constitute the best set of features for one of the available classifiers. For this reason, and to ensure that each input variable is present at least once in the population, the initial generation is seeded with one individual that contains all the input measurements, $F_1 = x_1, \ldots, F_d = x_d$, and uses the classifier that performs best on the full training set of the original data.

Exact duplicate individuals in the initial population waste space and reduce diversity (Koza, 1992*a*); no duplicate *features* are allowed in the initial population (therefore there can be no duplicate individuals either). The overall algorithm used for production of multi-tree individuals is shown in Figure 5.4. In short, the length of each individual is randomly selected according to a uniform distribution to be an integer between 1 and d inclusive. The sum of these lengths is calculated, M', and a population of M' unique features is created using the ramped half-and-half method. These features are then randomly sampled without replacement to form the multi-tree individuals. The classification algorithm for each individual is chosen as that of the set available which minimises the individual's training set error on the first N_o samples.

5.5 Mating Pool Selection and Fitness Evaluation

One of the main problems with EPrep is its computational burden: the number of training samples processed is proportional to $R \times G \times M \times n_{tr}$. The first three factors cannot be reduced, but it seems wasteful to use all n_{tr} training samples for the evaluation of each individual when some individuals will be such poor performers that their low worth is made evident using much fewer samples. In general, the minimum number of training samples required is different for each individual and for each generation. For instance, we would expect the need for more training samples to distinguish individuals in later generations as the population converges and individuals become more similar to one another than in initial generations. Although using fewer fitness cases can save on computation time, there are two pertinent issues:

1. how many fitness cases to use for each individual, and
2. which fitness cases to use.

The first issue is discussed here, while the second is dealt with in Section 5.6.

The issue of how many fitness cases are required for a genetic program has been tackled previously. The *Limited Error Fitness* method of (Gathercole and Ross, 1997*a*) uses a different number of samples for each individual. Fitness calculation is halted when the number of errors made by an individual exceeds an error limit. This way, time is not wasted on individuals that perform poorly. Another algorithm for selecting the number of samples per individual is the *Rational Allocation of Trials* (RAT) algorithm (Teller and Andre, 1997), which is described in some detail below. The RAT algorithm evaluates each individual until its rank is statistically unambiguous.

The RAT algorithm was chosen as the method for selecting the number of samples required for fitness evaluation because it is based on a statistical test. The RAT algorithm had to be significantly modified to be used with EPrep, as will be discussed shortly.

```
1. set $M' = 0$.
2. for $i = 1, \ldots M - 1$ loop
    (a) randomly select the number of features in individual $i$ with uniform
        probability from the range $[1, d]$: $L_i \sim U(1, d)$.
    (b) $M' = M' + L_i$.
   end loop.
3. Create $M'$ features $\{F_i\}^{M'}$ with randomly-selected output types using the
   ramped half-and-half method, ensuring that no two features are the same;
   $F_i \neq F_j \ \forall \, i, j = 1, \ldots M', i \neq j$.
4. Randomly shuffle the order of these features.
5. Set k = 1.
6. for $i = 1, \ldots M - 1$ loop
    (a) Set individual $i$ to consist of features $F_k, \ldots, F_{k+L_i-1}$.
    (b) $k = k + L_i$.
    (c) Initialise operator probabilities to $1/N_{ops}$.
    (d) Insert one intron between each pair of features.
    (e) Set the classifier label to refer to the classifier that results in the minimum
        training set error on the first $N_o$ samples pre-processed using individual
        $i$.
    (f) Insert individual $i$ into the population.
   end loop.
7. Create an individual containing only the original input variables,
        $F_1 = x_1, x_2, \ldots, F_d = x_d$
8. Set its classifier to be that which performs best on the original training set.
9. Insert this individual into the population.
```

Figure 5.4: The algorithm for creating the initial population.

5.5.1 The Rational Allocation of Trials

The RAT algorithm was introduced by Astro Teller and David Andre at the 1997 Genetic Programming Conference (Teller and Andre, 1997). The algorithm is essentially the BRACE algorithm (RACEing models using Blocking) applied to evolutionary algorithms. The RACE and BRACE algorithms were introduced in (Moore and Lee, 1994) as a method to speed up the selection of one from a large number of models. Given N data samples, the RACE algorithm refines the leave-one-out cross validation (LOOCV) error of N_{models} models in parallel, hence the name "race". When it becomes statistically unlikely that a model will win the race by having the lowest LOOCV error, it is removed from the race and no further computation time is wasted evaluating it. Under the strong assumption that the cross-validation errors are normally distributed, the mean and standard deviation of the error of model j based on k data points can be calculated:

$$\begin{aligned}\hat{\mu}_{kj} &= \frac{1}{k}\sum_{i=1}^{k} e_j(i)\\ \hat{\sigma}^2_{kj} &= \frac{1}{k-1}\sum_{i=1}^{k}(e_j(i)-\hat{\mu}_{kj})^2\end{aligned}$$

where $e_j(i)$ is the leave-one-out error of model j on sample i. The LOOCV error of model j based on k samples is $e^*_j = \hat{\mu}_{kj}$. Upon each iteration, model j is eliminated if there is another model j' that is statistically better or indistinguishable. This elimination process is shown graphically in Figure 5.5.

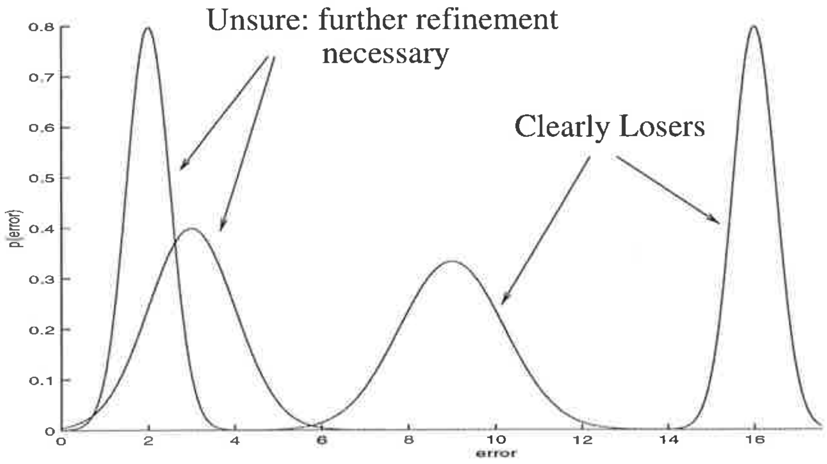


Figure 5.5: An example of the RACE model elimination process.

There are two parameters for the algorithm: δ is the statistical significance of the elimination test, and γ is a threshold (also a small positive number) which quantifies the maximum error difference between two models that are indistinguishable. For example, if $\delta = 0.001$ we have a one in a thousand chance of making an error in the test that model j has a mean error that is not less than that of some model j' by more than an amount γ. The rule is to eliminate j if there exists model j' such that:

$$Prob(e^*_j < e^*_{j'} - \gamma \mid e_j(1), \ldots, e_j(k), e_{j'}(1), \ldots, e_{j'}(k)) < \delta$$

The γ shift is introduced to stop similar models racing indefinitely, and therefore must quantify a trade-off between the consequences of erroneously eliminating a model, and the computational effort required to continue evaluation of the model.

The BRACE algorithm is an enhancement of the RACE algorithm using the statistical technique of Blocking (Box *et al.*, 1978). Blocking is the removal of variation in an experiment by keeping dependent samples together. In the RACE algorithm, for instance, there is a fair degree of dependence between the errors of different models for a given data sample: an outlier may be erroneously classified by nearly all models, but still contributes to the variance of each model error. If the errors coming from the same samples are kept together, variations coming from the data set are eliminated and the confidence intervals for the tests are reduced.

The race is now based on the difference of mean error rates of the models:

$$h^*_{jj'} = e^*_j - e^*_{j'}$$

The following statistics must be maintained:

$$\begin{aligned}\hat{\mu}^h_{jj'}(k) &= \frac{1}{k}\sum_{i=1}^{k}(e_j(i) - e_{j'}(i)) \\ \hat{\sigma}^h_{jj'}(k) &= \sqrt{\frac{1}{k-1}\sum_{i=1}^{k}\left[(e_j(i) - e_{j'}(i)) - \hat{\mu}^h_{jj'}\right]^2}\end{aligned}$$

Model j is removed when there exists another model j' such that:

$$Prob(h^*_{jj'} < -\gamma) < \delta$$

The improvement in performance brought by blocking can best be seen by examining the race between two identical models. The RACE algorithm will race these models until the confidence intervals of their error estimates becomes comparable to γ (which may never occur). BRACE, on the other hand, will quickly stop a race between identical individuals since the distribution of the differences between outputs will have zero variance.

There are three problems encountered when applying BRACE to evolutionary algorithms (Teller and Andre, 1997). Firstly, BRACE tests for the superiority of one model over others, whereas the roulette-wheel selection strategy requires that we know the quantity by which a model is better. Secondly, evolutionary algorithms typically select many individuals, whereas BRACE only selects one. The third complication is the choice of the γ value. In BRACE, the value of γ is selected based on the model's current performance. In an evolutionary algorithm, however, the quality of the *final* solutions is the pay-dirt, and an individual at some intermediate generation can contribute to this result in many various and subtle ways. For example, an individual may have a low fitness, but it may contain the genetic material that, through recombination, results in the best individual encountered.

The first two problems are addressed by using tournament rather than roulette-wheel selection. The tournament selection algorithm results in one clear winner per tournament, and does not need to know the absolute fitness values. The individuals (or models) must be evaluated only once per generation, so the tournaments need to be selected ahead of time. Then each individual is raced against all the other individuals in the same tournaments. The simultaneous performance of multiple tournaments is illustrated in Figure 5.6.

The third problem does not have a simple answer, as the method for selecting γ is not obvious. A trade-off must be sought between the computational cost of further evaluating a model, and the consequences of a model losing a tournament without statistical confidence[4].

(Teller and Andre, 1997) suggest several methods for selecting γ with varying complexity. The most basic thing to do is set $\gamma = 0$, which assumes that the cost of further trials is

[4] The decision will still result in the best individual based on the information we currently have, but that decision could have a low statistical significance.

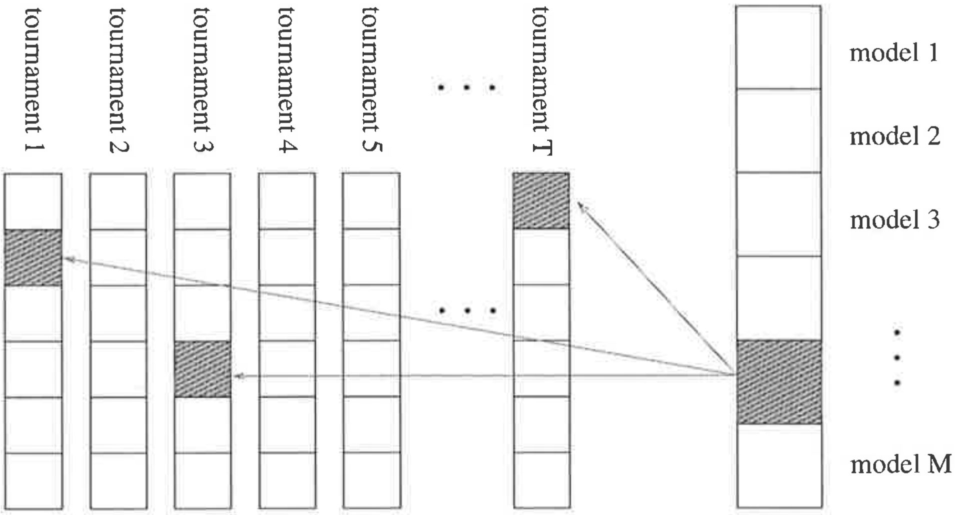


Figure 5.6: The simultaneous selection of tournaments using RAT.

insignificant compared to the utility of a tournament winner. For $\gamma = c$, a constant, the utility of a tournament winner is considered to be independent of its fitness. In general, an individual is considered more important for the evolutionary process if it has a higher fitness, and should therefore have a smaller value of γ to refine the comparison. The relative fitness of an individual can be quantified through the expected number of tournaments it should win based on its fitness rank. At the extreme level of detail, the whole myriad of complex factors that affect an individual's contribution to the final best solution could be considered.

The RAT algorithm performed each generation is shown in Figure 5.7. Note well that removing an individual from the contention list does not mean that the individual is of lesser use to the evolutionary algorithm: it simply indicates that no further investigation is required to ascertain this individual's relative performance. Indeed, an individual may be removed from the list because it won all its tournaments with clear statistical confidence.

Steps 5a and 5b in Figure 5.7 are not found in the original RAT algorithm; they were added so that models with larger observed mean error are removed first in comparisons between indistinguishable individuals, and to reduce the average number of comparisons. Further modifications to the algorithm were required, and are discussed next.

5.5.2 Modifications to RAT for EPrep

Three difficulties remain for the application of RAT to EPrep: how to update the fitness of an individual with each additional training sample, what distribution to assume for the classifier errors, and the choice of γ.

The saving in computational effort imparted by RAT relies on the computational complexity of the fitness evaluation being incrementally linear in the number of fitness cases. This property is defined here as the *computational independence* of the fitness function:

Definition 5.1 (Computational Independence) *Given two mutually-exclusive sets of fitness cases t_a and t_b, and some fitness function f_t whose evaluation is based on the cases in the set t, f is said to be* ***computationally independent*** *if:*

$$f_{t_a \cup t_b} = g(f_{t_a}, f_{t_b})$$

and $g(.,.)$ is some trivial function such as a linear combination.

1. Randomly select individuals for all M_{mp} tournaments, where M_{mp} is the size of the mating pool.
2. Initialise the contention list (Q) with all individuals.
3. Evaluate all individuals on the first N_o samples in the sample list S.
4. Set counter $j = N_o + 1$.
5. While $j \leq N$ and $|Q| > 0$ loop:
   (a) Sort the members of Q based on their current evaluation of fitness.
   (b) Visit each individual X in Q in decreasing order of error (*ie:* from worst to best).
   (c) Remove X from Q if, for every tournament t that X is in:
      - X is not in first place, and
      $$Prob(h^*_{XY_t} < -\gamma) < \delta$$
      or
      - X is in first place, and
      $$Prob(h^*_{Y_tX} < -\gamma) < \delta$$
      or
      - there are no individuals in t other than X.

      where Y_t is the individual from tournament t other than X that has the lowest error.
   (d) Evaluate all individuals still in list Q on training sample j.
   (e) Set $j = j + 1$.

   end loop.
6. Elect the individual in each tournament with the lowest error as the winner.

Figure 5.7: The Rational Allocation of Trials algorithm.

The implication of this definition is that the number of operations required to calculate the fitness for all available trials is roughly the same as that required to calculate the fitness separately for each of the individual trials and combine these sub-calculations to arrive at the overall fitness. The LOOCV error function used in (B)RACE is computationally independent, because the overall error estimate is a linear combination of the k leave-one-out errors evaluated so far.

For EPrep, fitness evaluation is a different story. We are not allowed the luxury of LOOCV error, since we cannot even afford to evaluate all training samples once. The objective function used in EPrep is not computationally independent because the addition of one sample to the training set requires the classifier to be re-trained and the predicted class labels to be re-calculated. If the computational complexity of the training algorithm is linear in the number of training samples, then the worst-case computational cost in evaluating an individual will be:

$$\begin{aligned} \mathcal{O}(N_o + (N_o+1) + (N_o+2) + \ldots + n_{tr}) &= \mathcal{O}((n_{tr} - N_o + 1)(n_{tr} + N_o)/2) \\ &\approx \mathcal{O}(n_{tr}^2) \end{aligned} \quad (5.2)$$

assuming $n_{tr} \gg N_o$. This is much more expensive than the worst case without RAT, which is $\mathcal{O}(n_{tr})$. The complexity of Equation(5.2) can be reduced by adding Δ samples instead of only one sample on each iteration (step 5d in Figure 5.7). The worst-case complexity of the fitness function then becomes:

$$\mathcal{O}(\frac{n_{tr}^2}{\Delta})$$

However there is a trade-off in choosing the size of Δ, since large values will more often result in worst-case complexity. Therefore to justify the use of RAT, the fitness function must be made computationally independent.

After consideration of several alternatives, the following compromise was reached. The classifier of each individual is trained on the first N_o samples only, since these must be processed for every individual anyway. This rough estimate of the classifier parameters is assumed to be adequate for generalisation, and may even promote the evolution of more robust features. After this initial training, the classifier remains static for the classification of future trials in the RAT algorithm. For the kth trial, the objective function of Equation(5.1) then becomes:

$$\begin{aligned} f_i(k) &= 100\% \times \sum_{j=1}^{k} \frac{I(h_i(\mathbf{y}_j) \neq c_j)}{k} \\ &= \frac{(k-1).f_i(k-1) + 100\%.I(h_i(\mathbf{y}_k) \neq c_k)}{k} \end{aligned}$$

which is computationally independent.

Since classification errors are binary (*ie:* correct or incorrect), the training error of each classifier is more appropriately modeled with a binomial distribution rather than with a Gaussian distribution. If the number of training samples is n_{tr} and the observed error rate is q, then:

$$\begin{aligned} \mu &= q.n_{tr} \\ \sigma &= \sqrt{n_{tr} q(1-q)} \end{aligned}$$

The binomial distribution can be approximated with a normal distribution when n_{tr} is large and q is not too close to one or zero; in practice, the approximation is good if $n_{tr}(1-q)$ and $n_{tr}q$ are both greater than 5.

There is no problem using this assumption for the RACE algorithm; the situation is more complicated for BRACE, since the distribution under test is that of the *differences* in binomial errors. This results in a trinomial distribution with possible values both-the-same, A-correct-B-incorrect and B-correct-A-incorrect. In order to keep things simple, the rather brazen assumption was made that the error differences are normally distributed. This assumption seems not unreasonable since if a binomial distribution can be approximated well with a normal distribution, then a trinomial distribution must not be far behind.

The final choice is in the selection of the γ value. The methods suggested previously all have a disadvantage in that they try to derive a value of error (γ) in terms of computational effort. It seems reasonable to call upon the ability of humans to effectively trade-off quantities of differing types. This can be done by asking the user a number of questions and using the information to compute values for γ. The process is complicated by the fact that we don't know how much computational effort will be required to further race two individuals: they may be disambiguated by the next fitness case, or they may require all of the remaining samples. Therefore a question must be framed as "how much error can be tolerated in discerning individuals rather than continue a race?". As in the previous methods, this will depend on the absolute fitness of the individuals undergoing comparison: a larger tolerance can be used for individuals with lower fitness than for highly fit individuals.

Fuzzy logic allows linguistic concepts to be manipulated quantitatively (Ross, 1995). Rather than asking the human a set of questions to determine the answer to a problem, the human's expertise can be incorporated into a fuzzy rule set. Considering that the error estimates e_i^* and e_j^* of two individuals can take on the linguistic values LOW, MEDIUM and HIGH, a table of fuzzy rules can be constructed to calculate the value of γ, as shown in Table 5.2.

Table 5.2: Fuzzy rule base for the γ function.

γ		e_i^*		
		LOW	MED.	HIGH
e_j^*	LOW	LOW	LOW	LOW
	MED.	LOW	MED.	MED.
	HIGH	LOW	MED.	HIGH

Each element of this table is interpreted as an IF-THEN-ELSE rule: for instance, IF e_i^* is LOW and e_j^* is HIGH, then γ is LOW. The lexical statements combined with an AND operation are called *conjunctive antecedents.* The outcome (following THEN) is called the *consequent.* The rules reflect the relative importance of highly fit (low error) individuals, since these individuals are more likely to have an impact on the quality of the final solution. Only one of the rules needs to be true, so they are combined with an OR function, and are therefore called a *disjunctive system of rules.*

The Mamdani (max-min) implication method of inference is used to calculate the crisp value of γ from the crisp values e_i^* and e_j^* (Ross, 1995). Both e_i^* and e_j^* have a membership μ in each of the antecedent fuzzy sets A_{LOW}, A_{MED} and A_{HIGH}. For each of the nine rules in Table 5.2, the conjunctive antecedents are calculated to derive the consequent fuzzy set. The consequents are then aggregated disjunctively, and the aggregated output is defuzzified using the centroid method. Using the minimum function as fuzzy AND, and the maximum function as fuzzy OR, the aggregated output membership function is:

$$\mu_B(y) = \max_{k=1,\ldots,9} [\min[\mu_{A_1^k}(e_i^*), \mu_{A_2^k}(e_j^*)]]$$

where A_1^k is the first antecedent of the kth rule, and A_2^k is the second antecedent of the kth rule. The defuzzified output value is:

$$\gamma = \frac{\int \mu_B(z) \cdot z \, dz}{\int \mu_B(z) dz}$$

Figure 5.8 shows visually the fuzzy sets and the operation of the min and max functions in the whole process of calculating γ for two rules. The actual membership functions to be used in EPrep are shown in Figure 5.9. In Figure 5.9(a), e_{min}, e_{ave} and e_{max} are the minimum, average and maximum classification errors in the population calculated based on the first N_o samples. In Figure 5.9(b), γ_{min} is the minimum value of the error tolerance, defined as the equivalent of one error from the k RAT trials:

$$\gamma_{min} = \frac{1}{k}$$

and:

$$\hat{\sigma}_{ij}^{h*} = \max\left\{\hat{\sigma}_{ij}^{h}, 2\gamma_{min}\right\}$$

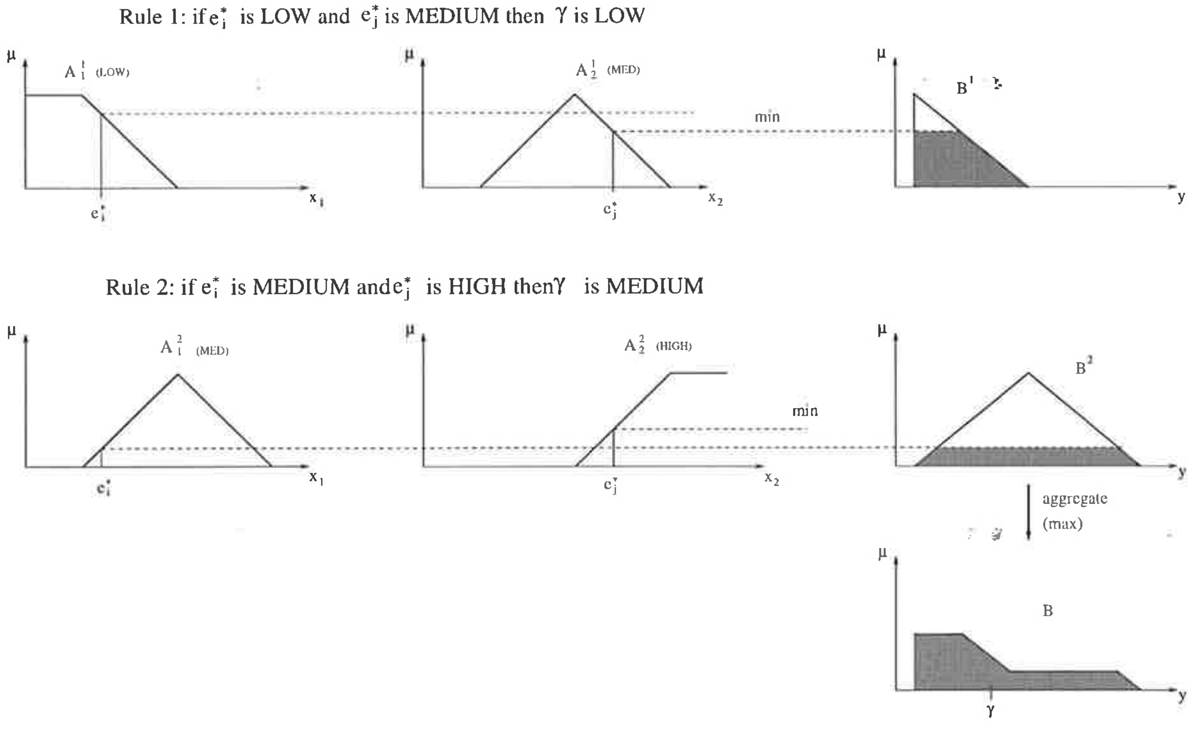
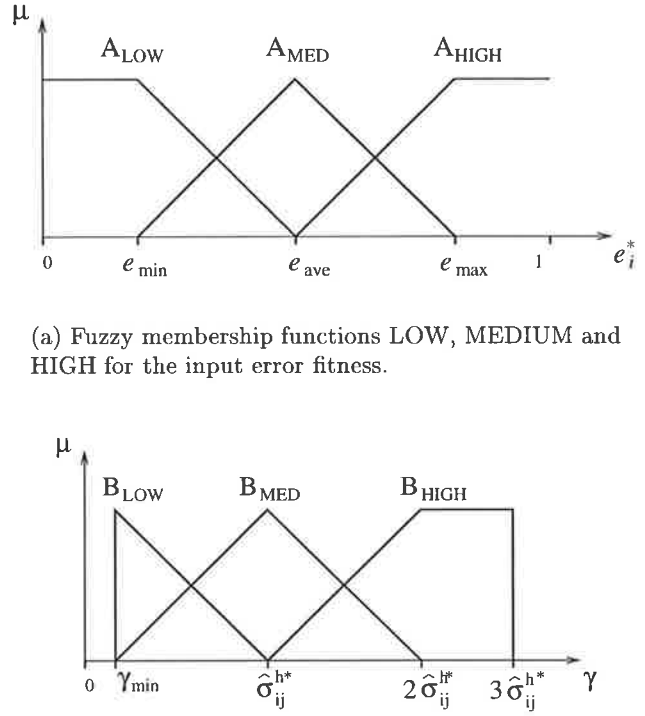

Rule 1: if e*ᵢ is LOW and e*ⱼ is MEDIUM then γ is LOW
μ
A¹₁ (LOW)
x₁
e*ᵢ
μ
A¹₂ (MED)
c*ⱼ
x₂
min
μ
B¹
y
Rule 2: if e*ᵢ is MEDIUM and e*ⱼ is HIGH then γ is MEDIUM
μ
A²₁ (MED)
e*ᵢ
x₁
μ
A²₂ (HIGH)
c*ⱼ
x₂
min
μ
B²
y
aggregate
(max)
μ
B
γ
y

Figure 5.8: Fuzzy inference of γ from e_i^* and e_j^*.

In Figure 5.10, the output of the fuzzy γ function is plotted for $\gamma_{min} = 0.1$, $\hat{\sigma}_{ij}^{h*} = 3.0$, $e_{min} = 5.0$, $e_{max} = 20.0$ and $e_{ave} = 10.0$.

5.6 Ordering of Training Samples

The question of which samples to use for training has been considered previously in the literature. Principled approaches have been taken to intelligently select the best training samples for the MLP (Plutowski and White, 1991; Zhang, 1994). In the approach of (Zhang, 1994), training of an MLP was started with $N_o < N$ samples and the sample size was increased by iteratively adding the sample that resulted in the largest error under the current



(a) Fuzzy membership functions LOW, MEDIUM and HIGH for the input error fitness.

(b) Fuzzy membership functions LOW, MEDIUM and HIGH for the model similarity tolerance γ.

Figure 5.9: Fuzzy membership functions for calculating γ.

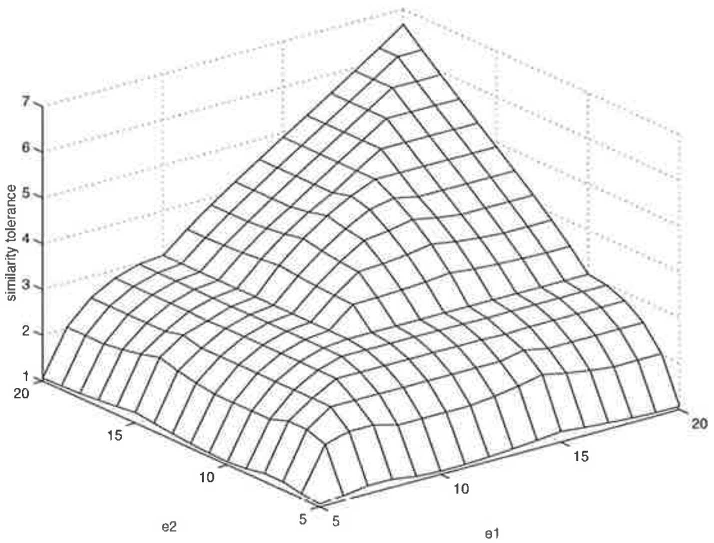


Figure 5.10: Similarity tolerance γ for $\gamma_{min} = 0.1$, $\hat{\sigma}_{ij}^{h*} = 3.0$, $e_{min} = 5.0$, $e_{max} = 20.0$ and $e_{ave} = 10.0$.

model; this equates to selecting the most difficult thing to learn next. Learning was found to be more robust than use of the whole training sample.

In the *Dynamic Sub-Set Selection* (DSS) method of (Gathercole and Ross, 1997*a*), a sub-set of cases is selected on each generation of a GP according to how difficult they were in previous generations and how recently they have been evaluated. Although the number of samples is chosen arbitrarily, the identity of the samples is biased towards samples that are difficult or have not been used recently. The Limited Error Fitness algorithm also re-orders the fitness cases in response to the performance of the best-of-generation individual. If there is no improvement in the BOG individual after several generations and the BOG individual makes fewer errors than the error limit, the limit is lowered and some of the easiest samples are shifted to the end of the list of fitness cases. The problem with both DSS and LEF is their dependence on several parameters which are difficult to choose.

The EPrep algorithm uses the first N_o samples to evaluate every individual, but thereafter the number of samples varies from individual to individual. The set of samples evaluated by each individual should be as similar as possible to minimise the amount of variation in the comparison of individuals. Therefore the sample order should be the same for each individual. According to this approach, the samples at the start of the list will be evaluated by all individuals, while those at the end of the list will be evaluated by only a few individuals. Therefore the sample order must be changed from generation to generation so that all samples are considered by the whole population at least once during evolution. Rather than totally re-ordering the training set each generation, only a few samples will be moved to the front of the list. This results in a relatively slight modification to the objective function, rather than the total upheaval associated with full re-ordering of the training samples.

A problem to be avoided with sample ordering is the sacrifice of certain classes because they have few training samples. For example, in a two class problem with 90 samples from class 1 and 10 from class 2, a classifier can obtain 10% error by misclassifying all samples from class 2 and getting all of class 1's samples correct. This individual would be ranked higher than a more even-handed individual that correctly classifies 80 samples from class 1 and 8 from class 2. To properly address this problem a cost matrix should be used. In EPrep the problem is partially addressed by ensuring that training samples are always sorted so as to preserve equal proportions of all classes. In cases where the classes are represented in the whole data set in unequal proportions, all the samples from the less-frequent classes may appear near the beginning of the training sample list.

According to conventional wisdom, the training sample order is updated by moving those samples which are the most difficult for the current population to the start of the list. There will be some samples at the end of the list that are not considered difficult simply because they have not been classified by many or by any individuals. Therefore the least-frequently visited samples are also periodically shifted to the front of the list. The question of which training samples to start with at generation 0 cannot be answered here, since at that stage there is no information about the problem. Therefore the algorithm of Figure 5.11 is used to initialise the order of the training samples. Note that the classifiers used by EPrep do not depend on the order in which the samples are presented.

If we keep the ordering the same for all generations, over-fitting may occur. If we totally re-order the samples after each n generations, the population could become unsettled so that convergence is slowed. As a trade-off between these two extremes, the algorithm of Figure 5.12 is used at the end of each generation.

This algorithm requires two counters to be associated with each sample: one for each visit, and one for the number of times it is erroneously classified. In steps 1a and 1b, if there are no samples left from class c then select from the class with the largest number of samples remaining. The algorithm ensures that a sample is moved to the start of the list (and is therefore evaluated by more individuals) if it has not been evaluated often or if the

```
1. Set list of samples, $S$, to be empty. Set $i = 0$.
2. While $i < n_{tr}$ loop
    • for $c = 1, \ldots, C$ loop
        (a) With uniform probability, randomly select a sample $s$ of class $c$
            without replacement from the training set.
        (b) If there are no samples left from class $c$, select $s$ from the class with
            the largest number of samples remaining.
        (c) Add $s$ to $S$.
        (d) Set $i = i + 1$.
        end loop.
   end loop.
```

Figure 5.11: Algorithm for initially sorting training samples.

```
1. for $i = 1, \ldots, n_{sort}/2$ loop
    • for $c = 1, \ldots, C$ loop
        (a) select the sample from class $c$ that has been least frequently classified
            and place it at the beginning of the list.
        (b) randomly select a sample from class $c$ in proportion to its classification
            error $E_j$ using roulette-wheel selection, and place it at the
            beginning of the list. The classification error of sample $j$ is:

            $E_j = \begin{cases} 0 & \text{if never classified;} \\ \frac{n_{error}(j)}{n_{classified}(j)} & \text{otherwise} \end{cases}$

            where $n_{error}(j)$ is the number of times sample $j$ has been incorrectly
            classified, and $n_{classified}(j)$ is the number of times sample $j$ has been
            classified.
        end loop.
   end loop.
```

Figure 5.12: Algorithm for re-ordering the training samples.

population is finding it difficult to classify on average. The disadvantage is that outliers which are irrevocably difficult to classify will spend more time at the beginning of the list, and over-fitting may occur.

The number of samples moved each generation is $C.n_{sort}$, so the magnitude of change each generation is controlled by the value of n_{sort}. A method of setting this value is to consider the desirable criterion that every sample is shifted to the beginning of the list at least once every g_{sort} generations, independent of its performance. Considering only the selection based on frequency of evaluation, it would take at worst $g_{sort} = (n_{tr} - N_o)/(C.n_{sort}/2)$ generations to cycle through all the samples. This yields a value of:

$$n_{sort} = \frac{2(n_{tr} - N_o)}{C.g_{sort}} \tag{5.3}$$

For typical values of $n_{tr} = 1000$, $N_o = 100$, $C = 4$ and $g_{sort} = 10$ we have $n_{sort} = 45$ samples.

Looking at things from the minimum disturbance principle perspective, n_{sort} should be chosen so as to disturb the learning mechanism only minimally. It is difficult to gauge the effect of changing some training samples on the error of a classifier, as the alteration of only a few samples can disproportionately affect the decision boundaries.

5.7 Optimisation of Constants

The efficient adaptation of constant values has been a point of difficulty in the GP community. While the gross structure of genetic programs can be improved through crossover, there remains no explicit mechanism for the optimisation of the constants, which can be considered as parameters for the overall model. Although the classifiers in EPrep act to some degree as a fine-tuning mechanism for the individuals, their effectiveness is limited to the adaptation of the decision boundaries: it is the parameters within the individual that determine the structure and separability of the data.

The problem of constants has previously been addressed through hybrid approaches. GP combined with simulated annealing (GP/SA) (O'Reilly and Oppacher, 1995*a*; O'Reilly and Oppacher, 1996; Sharman and Esparcia-Alcazar, 1993; Sharman *et al.*, 1995; Esparcia-Alcazar and Sharman, 1997) or GP combined with hill-climbing (GP/HC) (Harries and Smith, 1997; Iba *et al.*, 1994*b*; O'Reilly and Oppacher, 1994*a*; O'Reilly and Oppacher, 1994*b*; O'Reilly and Oppacher, 1995*a*; O'Reilly and Oppacher, 1996; Zhang and Mühlenbein, 1993). With these methods, the local optimisation algorithm is applied to each individual in the population with some frequency, or is integrated into the genetic operators to give selective recombination results. Some methods use a local optimisation that is particular to the representation (Hafner *et al.*, 1996; Sebag *et al.*, 1995; Iba *et al.*, 1993*a*).

Simulated Annealing (Press *et al.*, 1992) can be applied to an individual that contains at least one non-binary constant. A perturbation is applied to each constant in the individual, and the fitness is re-evaluated, f_i'. If the difference in fitness is positive, this point is selected as the new search point; if the difference is negative, the new point is selected with a probability:

$$e^{\frac{f_i' - f_i}{T}}$$

The temperature T is decreased according to the annealing schedule. Iteration continues until the objective is optimised or the maximum number of iterations is reached. The HC algorithm is similar, but an alternative solution is only accepted if its fitness is improved. SA is not used in EPrep since the evolutionary algorithm should place the constants in the neighbourhood of a local optimum, which then need to be adjusted with hill-climbing.

Two complications associated with using HC in EPrep are the computational effort of evaluating each alternative point, and choosing the scale of the perturbations. A successful hill-climb may require many iterations, and each iteration requires an evaluation of the fitness function. This means an increase in computational effort which should be compensated for by improved results for smaller populations, and convergence after fewer generations. As a compromise, only the first N_o samples are used to obtain a rough approximation of training set error for the local optimisation of individuals.

For any local optimisation method, the numerical perturbations must be concomitant with the constant being perturbed. There are no constants associated with boolean functions, so there are only constants of type Real and Enumerated. Since an enumeration's ordering does not have any semantic significance, a perturbation means uniformly selecting a new value from the enumeration. Enumerated values are therefore optimised according to the hill-climbing algorithm shown in Figure 5.13.

```
1. Let $\mathbf{x}(0) = [x_1(0), x_2(0), \ldots, x_n(0)]$ be the enumerated constants in the individual, and the set of values attainable by each be $A_1, A_2, \ldots, A_n$.
2. Set the number of iterations $k = 1$, and the set of previously-visited points $V = \{\mathbf{x}(0)\}$.
3. Evaluate fitness of starting solution, $y_0$, and set the best fitness and enumerated constant values to be this solution; ie: $y_{best} = y_0$ and $\mathbf{x}_{best} = \mathbf{x}(0)$.
4. while $k < \min(k_{max}, \prod_{i=1}^{n} |A_i|)$ loop:
    (a) loop:
          • for $i = 1, \ldots, n$ loop
              i. with probability $1/n$, modify the $i$th element of the current solution to be a new value from the set $A_i$; ie: $x_i(k) \in A_i, x_i(k) \neq x_i(k-1)$.
              ii. otherwise set $x_i(k) = x_i(k-1)$.
            end loop.
        while $\mathbf{x}(k) \in V$.
    (b) Evaluate $y_k$, the fitness of the individual with enumerated constants $\mathbf{x}(k)$.
    (c) $V = V \cup \{\mathbf{x}(k)\}$
    (d) If $y_k < y_{best}$ then set $y_{best} = y_k$ and $\mathbf{x}_{best} = \mathbf{x}(k)$.
    (e) k = k+1.
   end loop.
```

Figure 5.13: The Hill-Climbing algorithm for the optimisation of enumerated constants.

The mutation of real-valued constants is a more difficult issue, since the appropriate step-size for a given context is unknown. The *Simplex* optimisation method is used in EPrep to locally tune the real-valued constants in each individual. The method has the advantage that the step size adapts to the local landscape (Nelder and Mead, 1965). The problem is to optimise the fitness of an individual with respect to its n real-valued constants. This can be viewed as a search through an n-dimensional space. The simplex is the set of $(n+1)$ points (or individuals) $P_0, P_1, \ldots, P_n$ in the n-dimensional space. Let the fitness value of point (individual) i be $y_i = f(P_i)$, and define:

$$\begin{aligned} h &= \overset{\text{argmax}}{i=0,\ldots,n}\ y_i \\ l &= \overset{\text{argmin}}{i=0,\ldots,n}\ y_i \end{aligned}$$

That is, h stands for "high" and l stands for "low". $\bar{P}$ is the centroid of the points $P_i, i \neq h$. The algorithm is found in Figure 5.14. At each stage, P_h is replaced with a new point by one of three mechanisms: *reflection*, *contraction* and *expansion*. The new point P^* lies on the line joining P_h and $\bar{P}$. P^* lies on the opposite side of $\bar{P}$ from P_h by a distance determined by the *reflection coefficient* α ($\alpha > 0$), with $|P^*\bar{P}| = \alpha|P_h\bar{P}|$. If the new point P^* produces a new minimum, we can expand the step size by expanding the simplex. The *expansion coefficient*, γ ($\gamma > 1$) is the ratio of the distance $|P^{**}\bar{P}|$ to $|P^*\bar{P}|$, where P^{**} is the expanded point. If the new point P^* becomes the maximum of the simplex, the step failed and the simplex must be contracted. The *contraction coefficient*, β ($0 < \beta < 1$) is the ratio of the distance $|P^{**}\bar{P}|$ to $|P_h\bar{P}|$, where P^{**} is the contracted point.

Through this mechanism, the simplex is "walked" down-hill through the fitness landscape, adapting to local variations as it goes. The termination criterion used in EPrep is satisfied when:

$$\sum_{i=1}^{n+1} \frac{(\mathbf{y}_i - \bar{\mathbf{y}})^2}{n} < \delta_{conv}$$

where $\bar{\mathbf{y}}$ is the centroid of the simplex and δ_{conv} is a user-defined parameter.

If the local optimisation step is selected by the user, then it is applied each generation to every individual. In cases where the data are purely boolean, then there is no need for an optimisation step. In general an individual may contain real-valued and enumerated constants. In such a case, either the real-valued or enumerated constants are optimised, each with a probability of 0.5.

5.8 Genetic Operators

The most difficult task to perform when applying an evolutionary algorithm to a problem is selection of an appropriate representation and corresponding operators that will facilitate the search for fitter points in the solution space. There is a duality between operators and representation: anything that can be done in the representation can also be done in the operators. For instance, a GP tree can be represented as a tree or as a linear string in reverse polish notation: in either case, the genetic operators can be written to perform single-point sub-tree crossover. The action of the genetic operators on the representation determines the *transmission function*, which is the probability distribution of offspring from every possible recombination (Altenberg, 1994). The relationship between the transmission function and the fitness function determines the success of an evolutionary algorithm. If the transmission function is not geared towards fitter offspring, performance will be poor. Either the representation or the genetic operators can be modified to adapt the transmission function.

It is generally accepted that the crossover operator will not result in the proliferation of highly-fit building blocks unless the representation used provides building blocks in the form of sub-trees for the given problem. If we make the assumption that building blocks do exist in some form for the problem, then we can maintain the same representation and change the genetic operators to be appropriate for the given problem.

EPrep is intended to be applicable to a range of classification problems. We cannot generally assume that the standard single-point sub-tree crossover operator employed in GP will be appropriate for all such problems. Therefore an approach that suggests itself is to

```
1. Starting with constant values $P$, set $P_0 = P$ and generate the remaining
   $P_i, i = 1, \ldots, n$ by individually perturbing each constant:

   $$P_{ij} = \begin{cases} P_{0j} + \delta_j + \epsilon & \text{if } i = j \\ P_{0j} & \text{if } i \neq j \end{cases}$$

   where $\delta_j$ is a perturbation sampled from the normal distribution
   $\delta_j \sim N(0, 0.5P_{0j})$ and the addition of $\epsilon$ ensures a change when $P_{0j} = 0$.

2. Loop:
   (a) Calculate $y_i, \bar{P}, h, l$ and:
         $$P^* = (1 + \alpha)\bar{P} - \alpha P_h$$
   (b) Evaluate $y^* = f(P^*)$.
   (c) If $y^* < y_l$ then:
         i. Calculate:
              $$P^{**} = (1 + \gamma)P^* - \gamma\bar{P}$$
        ii. Calculate $y^{**}$.
       iii. If $y^{**} < y_l$ then:
              • Replace $P_h$ with $P^{**}$
            else
              • Replace $P_h$ with $P^*$
            end if.
       else if $y^* > y_i, i \neq h$ then:
         i. If $y^* \leq y_h$ then replace $P_h$ with $P^*$.
        ii. Calculate:
              $$P^{**} = \beta P_h + (1 + \beta)\bar{P}$$
       iii. Calculate $y^{**}$.
        iv. If $y^{**} > y_h$ then:
              • Replace all $P_i$'s with $(P_i + P_l)/2$.
            else
              • Replace $P_h$ with $P^{**}$.
            end if.
       else
         • Replace $P_h$ with $P^*$
       end if.

   until termination criterion is satisfied or maximum number of iterations is
   achieved.
```

Figure 5.14: The Simplex algorithm used for the optimisation of real-valued constants.

employ several complementary genetic operators, each of which may or may not be useful for the problem at hand. The following sub-sections describe the operators, the method for selecting them, and the use of introns in EPrep.

5.8.1 Self-Adaptation of Operator Probabilities

There are several ways in which the operators can be selected for deployment. The easiest is to use a static probability of selection for each operator. Slow convergence is expected with this naive method, since those operators which bring about the most improvement in fitness must wait for their turn, and the evolutionary process must continually compensate for the damage done by the inappropriate operators.

Another approach is to adapt the operator probabilities based on their observed performance for the problem. There are several ways of adapting strategy parameters in general; a taxonomy is given in (Hinterding *et al.*, 1997; Angeline, 1995). To summarise, there are three levels at which the adaptation can occur:

population: the adapted parameters belong to the whole population; for instance, the probability of crossover.

individual: the strategy parameters undergoing adaptation are specific to each individual.

component: the parameters are local to each gene of each individual in the population. For instance, the probability that a given bit will be mutated in a GA.

The mechanism for adaptation can belong to one of three categories:

deterministic: the strategy parameters are modified by a deterministic rule, and there is no feedback from the algorithm. For instance, the probability of mutation may decrease with time.

adaptive: the parameters undergo adaptation via a deterministic rule based on information fed back from the evolutionary algorithm (EA). For example, the "1 in 5 rule" for ES (Bäck and Schwefel, 1993): the ratio of successful mutations to all mutations should be 1/5. If it is greater, increase the standard deviation, if it is lower, reduce the standard deviation.

self-adaptive: the strategy parameters are evolved along with the population. The rationale is that parameters resulting in fitter offspring become associated with those offspring, and thereby permeate the population. An example is the mutation strategy parameters in evolution strategies (Fogel, 1995).

The goal of strategy parameter adaptation is for the algorithm to respond favourably to unexpected situations during evolution. The adaptation becomes more flexible as the level of adaptation becomes finer. Similarly, the mechanism will be more robust and able to cope with unforeseen situations the more adaptive it is. Therefore the most robust combination should be a self-adaptive component-level mechanism. It should be noted, however, that as the level of detail and adaptation increase, the burden placed on the EA increases also. This makes the complexity of the problem increase and may slow convergence. It is expected, however, that in difficult problems for which optimal solutions are difficult to find, the quality of the best solutions would be higher.

It is quite reasonable to assume that different operators will be more appropriate for different individuals on the same problem. Therefore the adaptation should be at least at the individual level. Ideally we would adapt the operator probabilities for each individual tree, but this would result in too many strategy parameters for an individual. As a compromise, a self-adaptive individual-level strategy is used in EPrep. A vector of operator probabilities is attached to each individual, as was done in (K. Chellapilla, 1997). The probabilities are initialised to be equal at generation zero. When an offspring is generated through recombination or mutation, it inherits the probability vector of its first parent. At every generation, the probability vector of each individual is modified by adding a Gaussian random perturbation to every element. Each element of the random perturbation vector is drawn from a normal distribution $N(0, 0.1/n_{op})$, where n_{op} is the number of operators used. Negative probabilities are set to 0.001, and probabilities that exceed 1 are set to 1, as in (K. Chellapilla, 1997). The probability vector is then normalised such that its elements have unit sum. When a parent is selected from the mating pool, roulette wheel selection is used to choose an operator according to the probabilities in the parent's operator probability vector. If the operator requires two parents, the next individual in the mating pool is used as the second parent. Since it was the first parent's vector that brought about the choice in operator, both offspring inherit the first parent's operator probabilities.

5.8.2 Operators and their Level of Adaptation

Given that a set of operators will be used, which operators are most appropriate? The operators must not be designed independently of the representation. Given the multi-tree representation used in EPrep (described in Section 5.2.1), variation can be conducted at two levels:

High-level Operators: which treat each tree like an indivisible gene, and therefore treat the genotype as a variable-length string of genes; and

Low-level Operators: which modify the trees themselves.

The set of high-level operators is the same as in a standard GA:

High-Level Crossover: analogous to single-point crossover used in GAs. If parent i has length L_i in features, $i = 1, 2$, then the range of valid crossover points is $[1, L_i - 1]$. A crossover point s_1 is randomly chosen between two features in the first parent with a uniform distribution. The crossover point in the second parent s_2 is also chosen randomly, but from a constrained set of points so that the offspring have length within the bounds $[1, d]$:

$$s_2 \in [\max(s_1 + L_2 - d, 1),\ \min(d - L_1 + s_1, L_2 - 1)]$$

These bounds are derived as follows. The lengths of the offspring are:

$$L'_1 = s_1 + L_2 - s_2 \quad (5.4)$$

$$L'_2 = s_2 + L_1 - s_1 \quad (5.5)$$

Given that $L'_1 \leq d$ and $L'_2 \leq d$, Equation(5.4) and (5.5) yield the inequality:

$$s_1 + L_2 - d \leq s_2 \leq d - L_1 + s_1$$

In practice the selection of crossover points is more complicated due to high-level introns, which are discussed in Section 5.8.3

Add-Feature Mutation: if $L < d$, a feature is created randomly using the grow method and inserted at a randomly-chosen point in the individual.

Delete-Feature Mutation: if $L > 1$, a feature is randomly chosen with a uniform distribution and deleted from the individual.

Inversion: randomly chooses two features in the individual and reverses the order of the features between them (inclusive). Inversion allows interdependent genes to come close to one another in the string, thus constructing short defining-length schemata which are less susceptible to destructive crossover (Holland, 1995). This operator is **not** applied to individuals in the mating pool to obtain offspring. Inversion is different from the other operators because it does not change the fitness of the individual to which it is applied; it is described below in Section 5.8.3.

Intuitively building blocks can exist at the high level of adaptation, since a good feature can to some extent contribute to the class-separability of the data independently of the rest of the features.

Low-level operators modify the tree-structured genetic material. The general utility of the standard single-point crossover operator is questionable since it is not always the case that single sub-trees are the building blocks for a problem (Angeline, 1997*b*; O'Reilly and Oppacher, 1995*b*). For example, the internal nodes of a sub-tree may be as likely to constitute a highly-fit code segment as the combination of code near the root and code near the bottom of the tree. Blind recombination should not be expected to work in every context. This state of affairs was addressed by the work on SMART operators (Teller, 1996) in which a population of operators was evolved that could arbitrarily recombine two individuals to form new offspring. The fitness of the operators was based on their ability to produce better offspring in the problem GP.

In the absence of knowledge about what form highly fit building blocks take for a problem, and of Teller's complicated system of operator evolution, the use of an arbitrary recombination operator cannot be justified. Therefore only mutation operators are used to modify the trees; this approach has been used previously in (Chellapilla, 1997; Angeline, 1996; O'Reilly and Oppacher, 1994*a*).

A mutation is applied by selecting a feature from the individual with uniform random probability and applying the operator to the selected tree. If the resulting tree contains no input variables, the process is repeated until the new one does. The following mutation operators are used in EPrep:

grow: randomly select a terminal node and replace it with a randomly-generated sub-tree.

oneNode: replace one node with another randomly-selected syntactically-legal symbol.

AllNodes: replace each node with another randomly-selected syntactically-legal symbol.

oneSymbol: replace each instance of a randomly-chosen symbol with one new randomly-chosen symbol.

swap: select a function node with two or more children having compatible return types, and swap two of the child sub-trees.

truncate: randomly select an internal node and replace it with a randomly-selected terminal.

hoist: select an internal node and replace the root node with the sub-tree from below that node.

All of the operators in EPrep restrict the maximum depth of the feature trees to a user-specified quantity. Note that bloating is unlikely to occur because sub-tree crossover is not used. The only mutation operators which can result in an increase in the size of an individual are the grow and add-feature mutators. However, bloating requires an increase in size associated with a constant fitness. This is an unlikely event for these two mutation operators, since the randomly-generated sub-tree is very unlikely to have no effect on the fitness of the individual.

In some situations the selected operator cannot be successfully applied to the parent individual. For example, the crossover operator may be applied to two parents with $L_1 = 1$ and $L_2 = 1$, or strong typing constraints may preclude the use of the oneNode mutator. When this is the case, the offspring returned is the same as the respective parent. To maintain diversity in the population duplicates are not allowed to be explicitly inserted: therefore the offspring are not allowed to be identical to the parents. If this does occur, the process of selecting a new genetic operator and applying it to the parent(s) is repeated until the offspring is (are) different from the parent(s). Finally, before insertion into the new population, each offspring is checked to see if it contains any duplicate features. If a feature is found to be an exact verbatim duplicate of another feature in the individual, it is deleted.

5.8.3 High-Level Introns and Inversion

Introns (as opposed to *exons*) are non-coding segments which do not contribute to the fitness of an individual. Introns have been observed and studied in biological genetics; a survey is found in (Wu and Lindsay, 1996). This inspiration from biology has been applied in evolutionary algorithms in the form of *explicitly defined introns* (EDI). EDIs are alleles that have no effect on an individual's fitness, but through their presence they modulate the probability of selection of a crossover or mutation point. Through the action of recombination and fitness-proportionate selection, the configuration of EDIs in an individual can adapt to protect co-adapted alleles against destructive crossover. They can therefore be considered as self-adaptive strategy parameters. EDIs have been used in genetic algorithms (Levenick, 1991) and genetic programming (Nordin *et al.*, 1995) with general success.

EPrep uses EDIs at a high-level, in between the trees. Note that EDIs are not allowed at the end of the string. To see how EDIs alter schema crossover probabilities, consider the individual in Figure 5.15. For a string of length L, there are $L-1$ distinct crossover sites. The probability of a crossover occurring at site i is $1/(L-1)$. Now if an individual contains T trees, and the number of EDIs between trees i and $(i+1)$ is $n_{EDI}(i)$ for $i = 1, \ldots, T-1$, then the probability that crossover occurs somewhere between tree i and $(i+1)$ is:

$$p(i) = \frac{n_{EDI}(i) + 1}{L-1}$$

Therefore as L increases and $n_{EDI}(i)$ decreases for tree i, the probability of adjacent trees becoming separated through crossover can become arbitrarily small. For example, there is a relatively low probability of separating F1 and F2 through crossover in Figure 5.15, so through natural selection and crossover, schemata of arbitrary defining length and high average fitness can proliferate and be protected from destructive crossover by intron accumulation.

Rather than physically representing the EDIs in the data structure of the individual, an integer is stored for each of the first $T-1$ trees which holds $n_{EDI}(i)$. Examine the crossover example in Figure 5.16. The length of the string is $L = T + \sum_{i=1}^{T-1} n_{EDI}(i)$. The crossover point j is randomly selected with a uniform distribution from the $L-1$ alternatives. The tree to the left of point j (call this tree i with immediately following crossover site j') is the end of the left crossover fragment, and the tree to the right (tree $i+1$) is the beginning of the right crossover fragment. When point l (and corresponding tree k with following crossover site l') is selected in the other parent, the first offspring has:

$$n'_{EDI}(i) = (j - j') + (n_{EDI}(k) - (l - l'))$$

and the second has:

$$n'_{EDI}(k) = (l - l') + (n_{EDI}(i) - (j - j'))$$

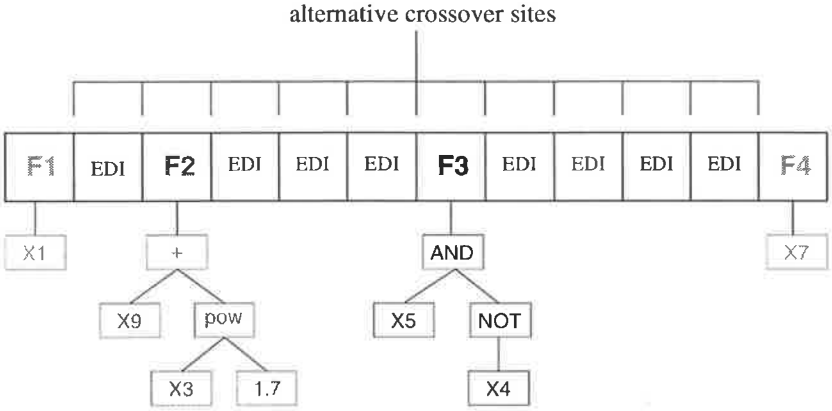


Figure 5.15: Explicitly Defined Introns in the multi-tree representation.

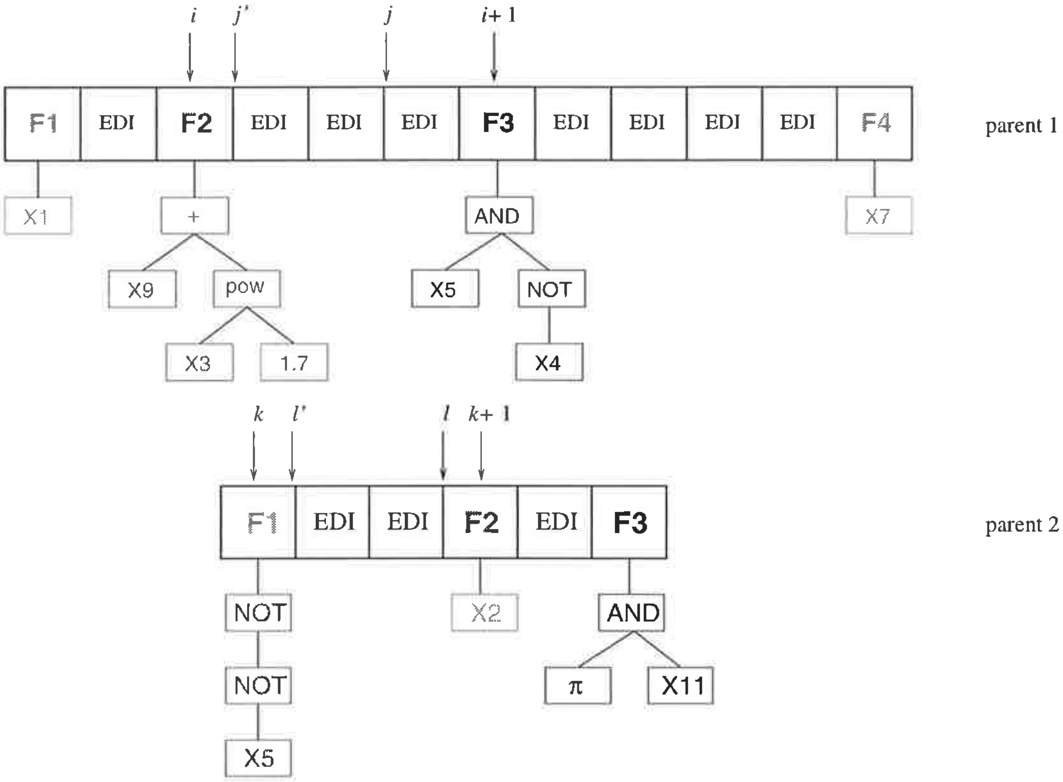


Figure 5.16: Example of crossover with explicitly-defined introns.

Introns can protect highly-fit short-defining-length schemata from destructive crossover, but cannot protect schemata of large defining length. For example, if an individual contains 10 features, and the first and last features are the main contributors to fitness, they will be easily separated through crossover to the detriment of the offspring. This problem has been recognised in genetic algorithms and is addressed by the *inversion operator* (Holland, 1995). Inversion selects a contiguous sub-string of the chromosome and inverts its order. An example is given in Figure 5.17. Considering our previous example, inversion with end points 1 and 9 can bring the first and last feature close together, making them less vulnerable to destructive crossover. Note that in genetic algorithms the meaning of each allele must be kept with the genes during inversion since the meaning of a gene is dependent on its position in the string, whereas in EPrep the utility of a feature is independent of its position.

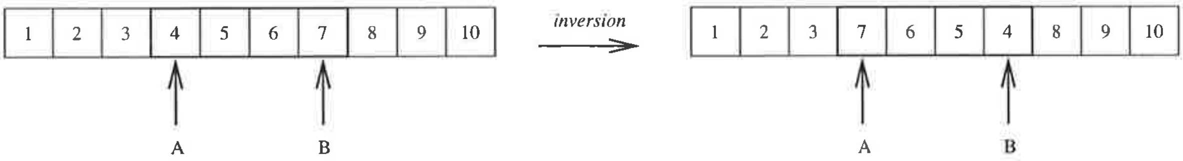


Figure 5.17: An example of the inversion operator.

The inversion operator is applied every generation to each individual in the population with a user-defined probability p_{inv}. Note that the inversion operator does not alter the fitness of an individual.

5.9 Stopping Criteria

Computational effort can be wasted by continuing converged or stagnated runs until the maximum number of generations is reached. The new stopping criterion could therefore be:

```
if (g = G) OR
    (validations set error + training set error = 0) OR
    (convergence criterion = TRUE)
then stop
```

A possible convergence criterion that tests for stagnation is:

$$STAG : f_{best}(g - n_{convergence}) - f_{best}(g) < \epsilon_{convergence}$$

where $n_{convergence}$ is a positive integer constant. This criterion is unlikely to fire for EPrep, though, since the training set error will fluctuate over generations for a single individual due to the samples being shuffled around. A more appropriate criterion is *training progress*, borrowed from (Prechelt, 1994):

$$TP_\beta : \text{stop after first generation } g \text{ with } P_k(g) < \beta \tag{5.6}$$

where $P_k(g)$ is the training progress:

$$P_k(g) = 100\% \cdot \left(\frac{\sum_{g'=g-k+1}^{g} e_{tr}^{BOG}(g')}{k \cdot \min_{g'=g-k+1,\ldots,g} e_{tr}^{BOG}(g')} - 1 \right)$$

for $g = k-1, k, \ldots, G$. $e_{tr}^{BOG}(g)$ is the lowest training set error at generation g, and k is the *training strip length*. Qualitatively, this criterion says to stop when the minimum training set error over the strip is not lower than the strip average by more than β percent. $P_k(g)$ is high when training set error is changing, and will approach zero in the steady-state as long as the training error does not oscillate.

There is another issue here, namely generalisation. Despite the steps taken to encourage parsimony, it was observed in preliminary studies (Sherrah *et al.*, 1997) that EPrep tended to over-fit, to the detriment of generalisation. A commonly-used method for stopping training before over-fitting occurs is *early stopping* (Prechelt, 1994). Some validation measure of the population can be monitored, and when it starts to increase consistently, the run is terminated. Note that the validation score can contain local minima, so one must be careful to allow the validation score some leeway to increase. The state at which the best validation score was obtained must also be held and reverted to upon termination.

Define *generalisation loss* at generation g to be (Prechelt, 1994):

$$GL(g) = 100\%. \left(\frac{e_{val}(g)}{e_{opt}(g)} - 1 \right)$$

where $e_{val}(g)$ is the validation measure at generation g and $e_{opt}(g)$ is the minimum validation measure before generation g:

$$e_{opt}(g) \overset{\min}{=}_{g' \leq g} e_{val}(g')$$

Qualitatively, the generalisation loss is the percentage amount by which the current validation measure exceeds the minimum achieved so far. As in (Prechelt, 1994), training ceases when the generalisation loss exceeds a pre-specified amount α:

$$GL_\alpha : \text{stop after first generation } g \geq k \text{ with } GL(g) > \alpha \tag{5.7}$$

Note that the GL criterion can only be true after one training strip; this allows for erratic behaviour of the validation measure at the beginning of a run. The validation measure $e_{val}(g)$ used in EPrep is the validation set error of the individual with the best training set error in the population (*ie:* the BOG individual). This quantity is calculated by pre-processing the training and validation sets with the individual, training the individual's classifier on all of the pre-processed training samples, and calculating the percentage misclassifications of the trained classifier on the pre-processed validation set.

The generalisation loss criterion, Equation(5.7), and the training progress criterion, Equation(5.6), are combined with an OR rule.

5.10 Cleaning the BOR Individual

The BOR individual must be *cleaned* by removing those features which do not contribute to the individual's fitness. This enables the fair comparison of the BOR individual's size with the size of the best-ever individual. The process starts with the full set of the individual's n features, $F(0) = \{F_1, F_2, \ldots F_n\}$. Each feature i is in turn removed from the BOR individual and its validation set error is re-evaluated. If the validation set error increases above the previous value, the feature is put back; otherwise, this feature is not necessary, and is left out of the individual. The process continues until all features have been checked. The algorithm is written in Figure 5.18.

5.11 Choice of Algorithm Parameters

EPrep is a complicated algorithm, and there are many parameters for the user to select. This section provides some guidance on how to select these parameters. It is an unfortunate truth that there are currently no methods for calculating the optimal parameters; indeed, the selection of parameters such as population size could constitute the topic of another thesis

```
1. Set $F(0) = \{F_1, F_2, \ldots F_n\}$.
2. Set $v_{min}$ to be the validation set error using $F(0)$.
3. for $i = 1, \ldots n$ loop
     • if $|F(i)| > 1$ then
         (a) Let $F(i) = F(i-1) - F_i$.
         (b) Calculate $v_i$, the validation set error using only the features in $F(i)$.
         (c) If $v_i > v_{min}$ then
               – set $F(i) = F(i-1)$
             else
               – set $v_{min} = v_i$
             end if.
   end loop.
4. Return the individual containing only the features in $F(n)$.
```

Figure 5.18: The algorithm for cleaning an individual's features.

altogether. The best parameters will generally be problem-dependent, and some helpful suggestions are provided here to guide the user in their selection. The following sub-sections deal with related parameters together.

Although this chapter has described the full EPrep algorithm, individual options can set through the graphical user interface, or through a text file.

5.11.1 Training Parameters

It is common for one classifier used by EPrep to perform considerably better on the original data than the other two classifiers; the ML classifier assumes this pole position most frequently. The user may want to eliminate this classifier from the set because it will be more difficult for EPrep to improve on its already good performance, resulting in features that are trivial or over-fit the data.

Assuming that the data have already been divided into training, validation and test sets, the number of initial samples N_o used by RAT and local optimisation must be selected, along with the number of samples to periodically move to the front of the list, n_{sort}. An expression has already been derived for n_{sort}, Equation(5.3), in terms of the turn-over time, g_{sort}, the worst-case number of generations required to shift each training sample to the front of the list. For convenience, that relation is repeated here:

$$n_{sort} = \frac{2(n_{tr} - N_o)}{C.g_{sort}}$$

The value of N_o decides the accuracy of the rough estimate of training set error used in RAT and local optimisation. It is these N_o samples which are used to estimate the true parameters of the simple classifiers. Therefore, we can frame the selection of N_o in terms of the degree of accuracy to which we wish to estimate the classifier's parameters. To simplify matters, we will only consider the class means, which are required by the ML and MDTM classifiers. The classifier operates on the features generated by some individual, $y_1, \ldots, y_g$.

When calculating the mean μ_{ij} of class i samples along feature j, the true mean will lie within the following confidence interval with probability $(1-\alpha)$:

$$\mu_{ij} - z\frac{\sigma_{ij}}{\sqrt{N_i}} \leq \mu_{ij} \leq \mu_{ij} + z\frac{\sigma_{ij}}{\sqrt{N_i}}$$

where $\text{Prob}(Z > z) = \alpha/2$, σ_{ij} is the true standard deviation of the population of y_j values from class i, and N_i is the number of samples in class i used to calculate the mean, $\sum_{i=1}^{C} N_i = N_o$.

It would be ideal to select N_i so as to shrink this confidence interval to some desired size for all classes; however σ_{ij} will be different for each individual, and it is unlikely that the y_{ij}'s will be normally distributed. Therefore all that can be practically done is to choose N_i so as to bring about some acceptable percentage reduction in the confidence intervals over the estimated standard deviation. For example, to shrink the confidence interval to width $0.2z\sigma_{ij}$, we require $N_i = 100$, and assuming equal class proportions at the beginning of the list, this makes $N_o = 100C$. If any wisdom can be gained from this analysis, it is that N_o *should be linearly proportional to the number of classes present.*

5.11.2 Evolution Parameters

The number of runs, number of generations and population size all interact to affect the likelihood of encountering the optimal solution to the problem. For example, running the algorithm for more generations increases the probability that a run will be successful. There is a point, however, after which it becomes more efficient to increase the number of runs rather than the number of generations. Similarly, increasing the population size will generally increase the probability of encountering the ideal solution on or before a given generation, but at some point it becomes more efficient to fix M and use more generations. These factors were considered in (Koza, 1992*a*) to measure the number of fitness evaluations required to solve a problem. Let $z = 1 - \epsilon$ be the probability of encountering the optimal solution, and $P(M, i)$ be the probability that the optimal solution is encountered on or before generation i of any given run. Then:

$$z = 1 - [1 - P(M, i)]^R$$

Note that $P(M, i)$ is monotonically non-decreasing in i. Re-arranging yields:

$$R(z) = \frac{\log(1-z)}{\log(1 - P(M, i))}$$

which is a non-linear dependence of the minimum number of runs on $P(M, i)$. The number of individuals required to obtain the solution, $R \times i \times M$, can be plotted to obtain the optimal number of runs and generations for a given population size.

The problem is that $P(M, i)$ can only be obtained by computationally-expensive statistical sampling, after which time the problem is solved anyway. The analysis was intended to measure the difficulty of a problem for GP, and not for the selection of optimal parameters. In practice, the number of runs should be as large as possible given the computational burden.

The size of populations in evolutionary algorithms has been a limiting factor in the field. While biological populations can be enormous, computational methods are usually limited to sizes ≈ 1000 by constraints on speed and memory resources. The Parsytec parallel processor used at Stanford University, consisting of 64 80-MHz Power PC 601 processors arranged in a toroidal mesh, can evolve an overall population size of 640,000 individuals representing analog circuits (Koza *et al.*, 1997). Although it is generally held that the larger the population size the better the performance, some results such as (Gathercole and Ross, 1997*a*) have exemplified the converse case.

The issue of the minimum-required population size has been addressed in (Goldberg *et al.*, 1991) for a genetic algorithm. There the statistical confidence of the *observed* ranking of the mean schema fitnesses for two schemata from the optimal solution is used to motivate the derivation of the general population-sizing relation:

$$M = 2z^2\kappa\frac{\sigma_\mu^2}{d^2}$$

where z is the cumulative distribution ordinate value at some level of confidence, κ is the number of competing schemata in the same partition as the two in question, d is the difference in mean fitnesses of the two schemata, and σ_μ^2 is the mean of the schema variances. Clearly it would be difficult to apply this formula in practice since the mean schema fitnesses for the optimal schemata are what we are trying to find in the first place.

The general advice on population sizing is to double the population size until no further improvement in performance is observed (Kinnear, Jr., 1994).

Tournament size has a direct effect on the selective pressure of the evolutionary algorithm: the larger the tournaments, the more possibility of involving the better individuals in the population. The probability with which individual i is selected for addition to the mating pool has been derived in (Bäck, 1994):

$$p_i = M^{-S}((M-i+1)^S - (M-i)^S)$$

where S is the tournament size and the individuals have been sorted in increasing order of fitness (for a minimisation task): $f(a_1) \leq f(a_2) \leq \ldots \leq f(a_M)$. The *take-over time* is a measure of selective pressure of a selection mechanism. It is defined as the number of generations required for the best-of-generation individual in the initial population to fill the population through the operation of selection only. For tournament selection the take-over time is:

$$\tau^* = \frac{\ln M + \ln(\ln M)}{\ln S}$$

Of course this is without the use of genetic operators, and so is only useful for comparisons between selection methods. Tournament selection provides much stronger pressure than roulette-wheel selection (Bäck, 1994). Higher selective pressure leads to a loss of diversity, which is offset in EPrep by the uniqueness of individuals inserted into the population. Nevertheless given the high selective pressure of tournament selection and the observation from initial experiments that EPrep tends to converge very quickly, the tournament size should be set as low as possible, $S = 2$.

The motivation for carrying the top N_{rep} individuals into the next population is to protect highly-fit individuals from destructive crossovers and mutations. A good individual may be selected for the mating pool many times, but there is a possibility that all of the offspring have considerably lower fitness, especially given the highly non-linear nature of the fitness function. Consider that for EPrep, the BOG individual is not necessarily the best generalising individual. Therefore if we could calculate how many individuals in the population are likely to be candidates for the best generalising individual based on their training set error, this would be an ideal value for N_{rep}, since the candidates are ensured propagation to the next generation where they may become the BOG individual due to the changing fitness function.

If we consider candidates for the best-of-generation generaliser to be those individuals which are statistically indistinguishable from the BOG individual, then we could calculate the confidence intervals for the BOG fitness assuming the errors take a Binomial distribution, and set N_{rep} to be the number of individuals falling within the confidence interval. This value, however, would change from generation to generation. At the beginning of a generation the population is more sensitive to the loss of good individuals, since the individuals become more and more similar as the population converges so that the loss of one good individual would not affect the average fitness much. For this reason and because the distribution of fitnesses at generation 0 does not depend on the value of N_{rep}, initial runs can be performed to calculate the average number of candidates at generation 0.

5.11.3 Termination Parameters

It has been observed in experiments that runs are almost always terminated by the generalisation loss criterion. Therefore the maximum number of generations should be set to some large amount to stop the program running forever, but no more notice need be paid to it. Similarly the training progress threshold is unimportant since the training error estimate is unlikely stagnate, given that the sample order is changing each generation. The more important parameters are the generalisation loss threshold α and the training strip length k.

Since the BOR individual will be chosen as that which obtains the minimum observed validation set error irrespective of the values of k and α, these parameters should be set such that a good result is observed. Ideally we would let the algorithm run forever ($k = \infty, \alpha = 100\%$) and choose the best encountered solution, but we realise that generalisation will be lost at an earlier point to a degree that makes us almost certain about the fruitlessness of further computation.

One could use the curvature of the validation error plot to look for the global minimum, but in practice the error can be quite erratic without any smooth trajectory. It may be wiser for the user to make a subjective judgment based on trial runs. Note that the GUI has an *Abort Run* button to allow the user to manually terminate the run.

The value of k imposes a lower bound on the number of generations performed per run, should the behaviour of the validation measure be volatile in early generations. A sensible approach would be to set $k = g_{sort}$, the turn-over time for the training samples. This way each training sample is considered by the whole population at least once.

5.11.4 Optimisation Parameters

It is safest to set the convergence value for the simplex algorithm to some small number, so that the algorithm stops when the variance of simplex scores is approximately zero. The choice of the maximum number of iterations is the main issue. For the simplex algorithm, $n + 1$ fitness evaluations are required just to initialise the simplex where n is the number of real-valued constants in the individual. For best performance the maximum number of iterations should be proportional to n, but for very large individuals in the population an inordinate amount of computation would be required. Therefore a constant value is used for all individuals.

There is a trade-off between the amount of time spent on local optimisation and on global optimisation: more iterations for the local optimiser may result in faster convergence to a solution, but require more computational power per generation. This state of affairs is similar to the discussion of number of runs versus number of generations: the use of optimisation increases $P(M, i)$ for a given generation i. Similarly the optimal trade-off can only be seen in retrospect, since the utility of the local optimisation is highly problem- and individual-dependent.

5.11.5 Representation Parameters

The choice of functions and terminals must be performed by the user, and depends on the data types and prior knowledge of the problem. The probability with which inversion is applied to individuals in the population is not critical to the performance of the algorithm, since the inversion operator does not directly alter the fitness of an individual. If the probability is set too high, short-defining length schemata will not have a chance to settle in the population. If too low, there will be insufficient modification of schema-defining lengths to be worthwhile. It must be taken into account that crossover will not be applied to every individual, since it must compete with the other available operators. Assuming that the probability of crossover associated with each individual does not converge to 1, p_{inv} should be set as high as possible (*ie:* $p_{inv} = 1$) to make use of every crossover that occurs.

Setting the maximum initial depth of features is tricky, since the optimal tree depth is problem-dependent. The overall search can be biased by the appropriate choice of this parameter, and a poor choice can lead to poor results. The only wisdom offered on this topic is to try different depths. The depth should be started at a small value rather than a large value, as computation time increases exponentially with the maximum tree depth and smaller solutions are preferable anyway.

Note that if the maximum initial depth is set to 1, each feature can only possibly be an input variable. In this case EPrep performs feature selection by evolving sub-sets of the original measurements; this is useful when there is a very large number of measurements available.

5.12 Conclusion

This chapter has presented a thorough description of the evolutionary pre-processor algorithm. The motivation for each design decision has been given, and heuristics for selecting appropriate parameters have been listed. The next chapter describes the experiments using EPrep to search for optimal pre-processors, and the corresponding results.

Chapter 6

Experimental Evaluation and Comparison

6.1 Introduction

In Chapter 4, the concept of a generalised pre-processor was introduced to facilitate a search for the optimal pre-processor for a given supervised classification problem. The evolutionary pre-processor described in Chapter 5 was developed to perform this search. It may be, however, that such a search is infeasible for practical problems, or is of no practical use because existing methods are adequate.

It is the purpose of this chapter to investigate the issue of whether a search for the optimal generalised pre-processor is a practical enterprise. There are two aspects to the practicality of the search: is it feasible, and is it worthwhile? The approach used here was to run the evolutionary pre-processor on several data sets from different problem domains. For each problem, the improvement in classification performance conferred by the evolved pre-processor was observed to assess the feasibility of the search.

To ascertain whether the process was worthwhile, a comparison was made to see if a generalised pre-processor could result in classification performance superior to conventional methods for supervised classification and automatic feature extraction.

This chapter begins with a description of the data sets used in the experiments. Next, the motivation for the experiments is given, and the experiments are described. The results of the EPrep algorithm on the test problems follow. The results of the other methods are then presented, followed by a comparison between them and EPrep. The conclusion presents a summary of the results.

6.2 The Test Problems

The methodology taken in this thesis was to examine a reasonably large set of appropriate problems and test EPrep's ability to learn from the data. Machine learning is of necessity an empirical science, and consequently analytical results are not the boast of this thesis. At our stage in history we do not have the tools to analytically decompose the dynamical behaviour of most machine learning algorithms. Although there are, fortunately, some analytical results for neural networks, genetic algorithms and other denizens of the machine learning literature, they rarely lend powerful insight since the algorithms had to be so grossly simplified that they scarcely resemble their original form.

The 15 data sets used in the experiments came from public domain repositories (White and Fahlman, 1993; Rasmussen *et al.*, 1996; Aviles-Cruz *et al.*, 1995; Meyer, 1996; ESPRIT, 1995; Merz and Murphy, 1996), and are briefly described in Table 6.1. A full description of each data set is given in Appendix C. Note that in this work, each data set is referred to by a unique name in bold-face, as recommended in (Prechelt, 1994); *eg:* **spirals** for the two intertwined spirals problem. Most of the data sets are real-world, while four are synthetic. Since many of the classification algorithms used in the experiments have a model selection step, the data were always divided into training, validation and test sets. The data sets were chosen to have the following desirable characteristics:

- Real, noisy data with possible missing or incorrect values.
- Medium to large size to avoid issues concerning the statistical validity of results due to small sample sizes. The number of samples is especially critical here because the data will be divided into the training, validation and test sets.
- Problems from different domains such as medical diagnosis, social science and image processing.
- Dimensionality of problems over a broad range.
- Number of classes of problems over a broad range.
- Non-trivial classification tasks: for instance, the well-known **mushrooms** (Prechelt, 1994) and **iris** (Aviles-Cruz *et al.*, 1995) problems were not used because they are almost linearly-separable, and present no challenge.
- Some enumerated attributes with a large number of distinct values, because some classification methods are sensitive to this parameter.
- Problems with a mixture of the attribute types real, enumerated and boolean.
- Varying difficulties of classification: the synthetic problems have an exact solution and there is no class overlap, while the real-world problems have varying degrees of class overlap.

The 15 data sets are plotted according to their dimensionality, number of samples and number of classes in Figure 6.1. From this figure it can be seen that the distribution of problem sizes is fairly uniform, with two outliers having a relatively large value along one or two of the axes. Note that there is no problem that is large according to all three dimensions.

Special characteristics attributable to some of the data sets are worth mentioning. The four synthetic databases are **balance**, **concentric**, **monks** and **spirals**. The **monks** data set consists of three separate problems defined on the same input domain. The **concentric** and **spirals** have the $x - y$ plane as their input domain, and therefore have a geometrical interpretation. The treatment of **spirals** used here is different to that of past studies because the data are divided into training, validation and test sets, and the input space is under-sampled in the training and validation sets. The **abalone** data set has a relatively large number of classes, and the **satimage** problem has a relatively high dimensionality and number of samples. The **australian** data set has an enumerated variable with a relatively large number of distinct values. The **titanic** domain only has 16 distinct combinations of the inputs, and is heavily over-sampled. Therefore classification is based on statistics collected from the data. It is expected that there is no discriminatory information contained in the **smoking** data set, and that little more can be done than to guess the most frequent class.

6.3 Test Descriptions

To investigate the question of whether a search over pre-processors is feasible, the evolutionary pre-processor was used to classify the data sets described in the previous section.

Table 6.1: Summary of data sets used in experiments. "Default Error" is the percentage of misclassifications obtained by always guessing the most frequent class. **monks** has 3 entries because it contains 3 different problems. "Dim." is the dimensionality of the data. Data types are real, enum(erated) and bool(ean).

Problem	Default Error(%)	No. of Samples	Dim.	No. of Classes	data types		
					real	enum	bool
abalone	83.51	4177	8	29	✓	✓	
australian	44.50	690	14	2	✓	✓	✓
balance	53.92	625	4	3	✓		
cmc	57.30	1473	9	3	✓	✓	✓
concentric	36.84	2500	2	2	✓		
diabetes	35.00	768	8	2	✓		
german	30.00	1000	20	2	✓	✓	
monks	50-32.9-47.2	432	6	2		✓	✓
satimage	76.18	6435	36	6	✓		
segment	85.71	2310	11	7	✓		
smoking	30.47	2855	9	3	✓	✓	
spirals	48.23	770	2	2	✓		
titanic	32.30	2201	3	2		✓	
vehicle	76.48	846	18	4	✓		
yeast	68.80	1484	8	10	✓		

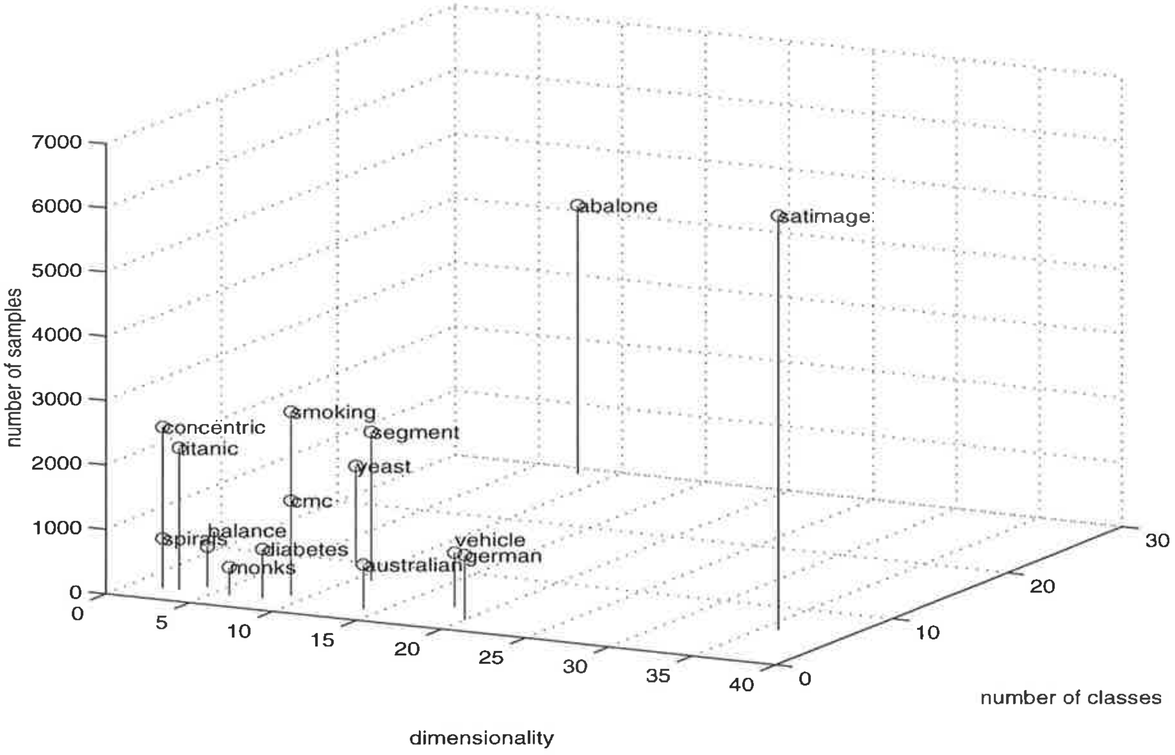


Figure 6.1: The 15 data sets plotted according to their dimensionality, number of classes and number of samples.

On the basis of these results, a comparison was then performed with existing methods to investigate whether the new approach is worthwhile. While addressing these two fundamental issues, other potential questions about the use of EPrep as a classification method were investigated. The following questions could potentially be asked of EPrep, or of any machine learning algorithm:

1. Does it work at all?
2. For what sort of problems is it appropriate? What are the limits on problem size and data type?
3. How long does it take to run?
4. If it is a stochastic algorithm, how much variation is there in the results?
5. Do the results yield useful information about the problem? Is the information reliable, or does it vary too much over runs to be useful?
6. Does it learn efficiently?
7. How does it compare with other supervised learning algorithms, and what advantages does it have?

For the practitioner, these questions can all be summarised into one: "why should I use this algorithm?" Simply running EPrep on a broad range of example problems constituted sufficient experimentation to address the first 5 of these questions. The last question was addressed through a comparison with other methods. The question of efficiency depends on the criterion of interest. If, for example, search efficiency is the issue then the no-free-lunch methods mentioned in Section 3.2.3 may be of use, or a comparison with hill-climbing or random search may serve as a suitable benchmark. If, on the other hand, classification accuracy is paramount, then the user would not care if an algorithm took longer to run as long as it produced a more efficient classifier. Other criteria are the computational complexity of the algorithms, the number of input measurements required by the trained classifier, and average number of computations per classification of the trained classifier. The efficiency comparison performed in these experiments was based on computational complexity.

The next sub-section describes the general decisions and conditions for the experiments. The succeeding sub-section is devoted to the experiments applying EPrep to the test bed, followed by a sub-section describing the comparative experiments.

6.3.1 Experimental Design

In the experimental design, the results were analysed statistically where possible rather than naively jumping to conclusions. There were several sources of variation in the experiments which were addressed to make the results more sound:

problem type A result or phenomenon that arises for one problem may not for another problem. To address this, different problems from different domains were used.

stochastic algorithms Those algorithms relying on random initial conditions or having a stochastic element during training were run 10 times each. This number of runs was chosen as a compromise between statistical accuracy and computational cost.

partitioning of the data Classification errors can be heavily dependent on the partitioning of the data into training, validation and test sets. For example, if a random partitioning happens to result in a training set that gives little information about the test set, then tests for generalisation will yield poor results. To offset this problem, the experiments were performed separately for three different permutations of each data set. This number of partitions was selected to give a feel for the effect of permuting the data on the results, while keeping the amount of data generated manageable. Each permutation of a data set is referred to by the data set name augmented with its number; *eg:* the second permutation of the two spirals data set is referred to as **spirals2**. The data were permuted randomly while keeping the same relative class proportions in each of the training, validation and test sets.

One of the issues in preparing the data was what proportions to use for the training, validation and test sets. There is a trade-off because we would like to use all of the data for training to obtain the most accurate model, but we would also like to use as much as possible for testing to obtain an accurate estimate of the misclassification rate. Fortunately in this case the results do not rely heavily on this choice, because all of the experiments are comparative in nature. Therefore it is not so much an issue of whether the classifier learned the structure contained in all of the data from the problem, but rather which of the classifiers managed to learn the most from the data used for training. With this in mind, all experiments used a partition of 50% training, 25% validation and 25% test. The exact number of samples in each set is shown in Appendix C. Note that there are two exceptions to this rule, **monks** and **spirals**. In the case of **monks**, the data were already partitioned into training and test. The training samples were randomly divided into a training set and a validation set. In the case of **spirals**, the training samples were randomly partitioned into the training and validation sets for each of the three permutations.

In the experiments involving multiple runs of stochastic algorithms, the "best" run was used as the measure of the overall algorithm performance. It is very important to note, however, that **best does not mean that run with the minimum test set error**. If the test set is used to select the best classifier, then it automatically becomes part of the training data. Rather, the run with the minimum validation set error is selected as the best run. If a classifier has good generalisation capabilities, then the test set error will be correspondingly low.

6.3.2 Analysis of EPrep

The experiments were conducted on EPrep by performing 10 runs on each of the 3 permutations of the 15 data sets: 450 runs overall. Each run resulted in a best-of-run individual, consisting of a pre-processor and a classifier. This individual can in itself be considered as a non-linear classifier: it can be applied on its own to the classification problem without ever having to run EPrep again. Therefore a before-after comparison was made between the *classifier on the raw data* and the *classifier on the pre-processed data.* The comparison between these two classifiers indicated whether the performance of the classifier could be improved upon by finding an appropriate generalised pre-processor. The misclassification errors of the two classifiers on the test set were compared using McNemar's test (see Section 2.11.1). This test is built into EPrep, and the probability that the evolved pre-processor has improved classification performance is displayed in the HTML reports generated. Note that this test does not tell us by how much EPrep has improved the performance of the classifier.

Referring back to the questions asked in Section 6.3, trialing EPrep on different data sets helps to answer the questions of whether the search for a generalised pre-processor is feasible, what sort of problems are best solved using this method, and what the limitations on problem size are.

After the best results were examined, the variability in the results was analysed to explore the likelihood of obtaining the best solution were the algorithm run again with a different random seed. The variability in the structure and content of the pre-processors themselves is also an issue, and is important for ascertaining the usefulness of EPrep for knowledge discovery. To casually examine computational complexity, the average execution time of the algorithm for each data set was recorded. These execution times are not directly useful for comparative purposes, because they depend on the platform used to run the algorithm. The execution times is included only to provide an order of magnitude estimate of the time taken for the algorithm to solve a problem. The computational complexity of the EPrep algorithm is examined in Section 6.8.4.

The parameters for EPrep were selected according to the guidelines given in Section 5.11. In some cases, the parameters were iteratively refined by changing parameters that were obviously chosen poorly. The following notes summarise the main points of the parameter selection process:

- The parameters used were different for each problem, but usually kept the same for each of the three permutations of the same problem. The exceptions were **cmc**, **german**, **monks**, **satimage** and **yeast**.

- The power function $\mathrm{pow}(a, b) = a^b$ was avoided because it was found to produce poor results due to its highly non-linear behaviour.

- For many of the data sets, one of the three classifiers used by EPrep (see Section 5.2.3), MDTM, PPD and ML, was excluded from the set used by EPrep because it performed significantly better than the other two classifiers. This "tall poppy" was often the ML classifier. The classifier was excluded in these cases because it was doing most of the work in classification, leaving little for EPrep to do. It was found that if the classifier was left in, there was too little room for improvement on classification error for EPrep to evolve useful features. Removing the best classifier put more pressure on EPrep to evolve features that separate the data, yielding more interesting results. The exceptions to this rationale were **monks2** and **vehicle** for which EPrep was unable to achieve a reasonable error rate without the ML classifier. It may be that excluding the most accurate classifier is a better approach for knowledge discovery, but if accuracy is the main objective then the best classifier should perhaps be left in the set.

- In some cases, the parallelepiped classifier was excluded from the set used by EPrep because it did not generalise well with the evolved features.

- Local optimisation was not used for **monks3** and **yeast** due to restrictions on computation time, and the limited benefits of its use.

6.3.3 Comparison with Other Methods

To examine the strengths and weaknesses of the EPrep methodology, a comparison was performed with 7 other classification methods. Without the benefit of experimental results, EPrep has some advantages over other supervised classification methods:

- the search algorithm can escape local optima;

- the features generated are true for all time, and can be interpreted by the user;

- a range of different non-linear and discontinuous functions can be combined to construct the decision surfaces;

- the data dimensionality is reduced and the classification algorithms are simple, so the solution can be implemented inexpensively; and

- the input measurements that are needed for classification are automatically selected.

How useful these advantages are in practice depends on the application at hand.

In terms of classification performance, a comparison with established methods was performed. For each of the problems listed previously, EPrep's test set classification errors were compared with those of the following methods:

- Multi-Layer Perceptron (MLP) trained with the RPROP algorithm; see Section 2.7.2
- Decision Trees (QUEST); see Section 2.8.3
- Generalised Linear Machine (GLIM); see Section 2.7.1
- k-Nearest Neighbours (kNN); see Section 2.6.4
- Parallelepiped Classifier (PPD); see Section 2.3.1
- Minimum-Distance-to-Means Classifier (MDTM); see Section 2.6.2
- Gaussian Maximum Likelihood Classifier (ML); see Section 2.6.3

The MLP was chosen as part of the comparison because it is another automatic feature extraction method, and is widely used. The QUEST algorithm was chosen because, like EPrep, it generates solutions that can be scrutinised by the user. The other methods are fairly simple and were used because they were readily available (PPD, MDTM and ML are intrinsically part of EPrep), and can be used to indicate problems for which complicated decision boundaries are not necessary.

The statistical test used to compare the best-case performance of the classifiers is the one-way repeated measures design described in Section 2.11.3. This method gives pair-wise confidence intervals for all the algorithms. Comparing EPrep with these other methods indicates in what situations the generalised pre-processor approach is advantageous.

The average-case performance of EPrep was compared with that of the MLP. The t-test described in Section 2.11.2 for the difference between means of distributions with unequal variances was used to compare the mean test set errors over the 10 runs performed. The understandability of the solutions generated by EPrep was compared with QUEST's decision trees to investigate the utility of EPrep for knowledge discovery. A final comparison was made between the execution times and computational complexities of the classification algorithms. The execution times are only recorded to give the reader an idea of the time required for each algorithm, and cannot be used for a direct comparison due to the multiplicity of platforms used.

There is often an intrinsic bias in comparisons between a new algorithm and existing algorithms, because the person conducting the experiments is usually the author of the new method. Therefore the experimenter is the expert on how to select parameters for the new method, but does not know how best to configure the existing methods. The comparison made in this thesis was intended to be as unbiased as possible. Although for QUEST and the MLP the same parameters were used for all data sets, different parameters were used if it was obvious that the defaults were not appropriate. The parameters for EPrep were selected according to common sense, and in some cases iteratively refined by changing parameters that were obviously chosen poorly. The iterative process was limited, however, so that sub-optimal parameters were also used for EPrep.

The experimental particulars for each of the methods are set out below.

6.3.3.1 Multi-Layer Perceptron

The experimental set-up for the Multi-Layer Perceptron (MLP) was virtually identical to that used in the Proben1 technical report (Prechelt, 1994). An initial search over 12 network architectures was performed to find the best topology, termed the *pivot* architecture. The pivot network was then run 10 times to obtain the final results. The MLP algorithm was implemented in Matlab.

One output layer neuron was used for each class, and the target vectors were constructed using the 1-of-m encoding (see Section 2.7.2). In order to improve training performance, the target vector values $0, 1$ were shifted to $0.1, 0.9$. The networks contained all connections between consecutive layers. The neuron activation function was the sigmoid:

$$f(x) = \frac{ax}{1 + a|x|}$$

with derivative:

$$f'(x) = \begin{cases} a & \text{if } x = 0; \\ \frac{f^2(x)}{ax^2} & \text{otherwise.} \end{cases}$$

a is the slope at the origin; a value of $a = 1$ was used. This sigmoidal function was used in preference to the traditional tanh function because it does not require calculation of an exponential function. The RPROP algorithm (see Section 2.7.2.2) was used to train the networks. The GL_α and TP_β stopping criteria (see Section 5.9) were used to halt training with training strip length k and a maximum number of 3000 epochs.

The 12 one- and two-hidden-layer network architectures trialed were 2+0, 4+0, 8+0, 16+0, 24+0, 32+0, 2+2, 4+2, 4+4, 8+4, 8+8, 16+8, where the notation x+y means x neurons in the first hidden layer and y in the second hidden layer. For each architecture, two runs were performed using linear output neurons and one using sigmoidal output neurons, denoted x+yl and x+ys respectively. Therefore a total of 36 initial runs were used to choose the MLP architecture. The pivot architecture was chosen using the following algorithm, recommended in (Prechelt, 1994) for use with early stopping. The set of networks with validation set classification error (not sum-squared error) rates within 5% of the lowest were chosen as competitors. From these, the network with the largest number of weights was chosen. If two competitors had the same size, the one with the smaller validation set error was chosen unless the linear architecture appeared twice, in which case it was chosen.

The parameter settings used to find the pivot architectures were:

$$\begin{aligned} k &= 50 \\ \alpha &= 5 \\ \beta &= 1.5 \\ \eta^+ &= 1.1 \\ \eta^- &= 0.5 \\ \Delta_{max} &= 50 \\ \Delta_{min} &= 0 \\ \Delta_{max}(0) &= 0.2 \\ \Delta_{min}(0) &= 0.05 \\ \omega_{min} &= -0.5 \\ \omega_{max} &= 0.5 \end{aligned}$$

Refer to Sections 2.7.2.2 and 5.9 for an explanation of each parameter.

The pivot architecture was then trained 10 times to obtain a best and average classification error on the test set. The parameters used in these final runs were:

$$\begin{aligned}
k &= 5\\
\alpha &= 5\\
\beta &= 1.5\\
\eta^{+} &= 1.2\\
\eta^{-} &= 0.5\\
\Delta_{max} &= 50\\
\Delta_{min} &= 0\\
\Delta_{max}(0) &= 0.02\\
\Delta_{min}(0) &= 0.005\\
\omega_{min} &= -0.01\\
\omega_{max} &= 0.01
\end{aligned}$$

except for the **concentric** problem, for which the architecture-finding parameters were used.

Since data are usually pre-processed specifically for neural networks (see Section 2.7.2.1), the following modifications were made to all data files:

- Each enumerated variable was replaced with a set of boolean values, one for each value in the enumeration. More formally, for a variable x_i which can take on one of n distinct values:

$$x_i \rightarrow [x_{i1}, x_{i2}, \ldots x_{in}]$$
$$x_{ij} = \begin{cases} 1 & \text{if } j = x_i, \\ 0 & \text{otherwise} \end{cases}$$

- Real variables were scaled to the range $[0, 1]$.

This process resulted in an increase in the dimensionality of the data. The old and new dimensionalities for each problem domain are shown in Table 6.2.

For the MLP, both the best and average classification errors were compared with those of EPrep.

6.3.3.2 QUEST

There are many different decision tree methods. QUEST (see Section 2.8.3) was chosen for this comparison because it was recommended as the best overall decision tree method from a survey of 22 decision tree methods on 32 data sets (Lim *et al.*, 1997), and the software is public domain (Loh and Shih, 1997*a*). QUEST version 1.6 was used on the PC for most of the data sets, but version 1.7 was used on Unix workstations for the larger problems. In each case, the default parameters were used, listed in Table 6.3. The training and validation sets were combined into a single learning sample, and the test set was kept aside as a hold-out set.

6.3.3.3 *k*-Nearest Neighbours

The *k*-Nearest Neighbours algorithm uses the full set of training samples to determine the class of a newly-presented sample. When a new sample is presented, the distance from it to

Table 6.2: Dimensionality of the problem domains before and after preparation for the MLP.

Problem	original dimensionality	MLP dimensionality
abalone	8	10
australian	14	40
balance	4	4
cmc	9	12
concentric	2	2
diabetes	8	8
german	2	63
monks	6	15
satimage	36	36
segment	11	11
smoking	9	22
spirals	2	2
titanic	3	8
vehicle	18	18
yeast	8	8

Table 6.3: Default parameters used for QUEST simulations.

Parameter	Value
priors	estimated from data
misclassification costs	equal
minimal node size	5
splitting method	univariate
variable selection method	(unbiased) statistical tests
alpha value	0.05
method of split point selection	discriminant analysis
number of SEs for pruning	1.0
pruning method	10-fold cross-validation

all the training samples is calculated, and the k nearest samples are used to decide the class of the sample. The decision is made by majority vote. In the case of a tie, the class whose samples have the lowest aggregate distance is chosen. The distance function used depends on the application; in this case, Euclidean distance was used.

The hyper-parameter k was chosen by minimising the validation set error over the range $k \in 1, \ldots N_{tr}$. This value of k was then used to classify the test set using the training and validation sets as the reference data. Note that for **satimage**, only k values in the range [1, 1300] were examined due to excessive computation times, and the steady progression toward larger errors for higher values of k.

6.3.3.4 Generalised Linear Machine

The generalised linear machine, described in (Nilsson, 1993), is similar to the linear perceptron. It separates the feature space with C hyper-planes, which are positioned via iterative training with an error-correcting rule.

The neuron weights were always initialised to zero, so that the results were independent of the initial conditions. The parameter that could vary was the number of epochs, which is the number of times the whole training set was presented to the classifier. During training, the minimum training set error indicated the best set of weights. To avoid over-training, the algorithm was run for each number of epochs in the range $[1, 1000]$, and the classifier with the minimum validation set error was chosen as the best.

6.3.3.5 Minimum Distance to Means Classifier

This is a very simple classifier indeed. The mean of the training samples is constructed for each class. When a new sample is classified, it is assigned to the class of the mean that is nearest. Euclidean distance was used here. There was no need for validation, so the training and validation sets were merged into a single training set.

6.3.3.6 Parallelepiped Classifier

One step up from MDTM, the parallelepiped classifier calculates the extents of the training samples from each class to form an enclosing parallelepiped. A novel sample that falls in a single parallelepiped is assigned the class of that parallelepiped. If the sample falls in more than one parallelepiped or is unclaimed, the centres of the parallelepipeds are calculated and the algorithm reverts to minimum-distance-to-means. Again, the training and validation sets were merged to form a single training set.

6.3.3.7 Gaussian Maximum Likelihood Classifier

This classifier assumes a single Gaussian density function for each class, resulting in quadratic decision boundaries. There is no model selection step, so the training and validation sets were merged into a single training set to estimate the means and covariance matrices for each class.

6.4 Results of EPrep

It is natural that, since this thesis is about EPrep, more attention should be paid to its results than to those of the other methods. The summarised results of the EPrep runs are shown in Table 6.4. The meanings of the columns in the tables are:

Problem: the name of the data set and permutation.

Table 6.4: Classification results of EPrep algorithm.

Problem	tr.	val.	test	Δ	$1-\alpha$	test%		no.	no.	classifier	time
	error%	error%	error%	error		μ	σ	ftrs	nodes		h:m
abalone1	73.71	73.18	75.60	**14.64**	1.00	76.81	1.50	4	91	MDTM	42:55
abalone2	72.51	73.37	73.97	**18.09**	1.00	76.62	1.29	2	42	MDTM	11:00
abalone3	73.23	73.56	74.64	**16.84**	1.00	77.51	2.01	6	102	MDTM	10:58
australian1	17.97	10.47	10.40	**27.17**	1.00	11.39	0.37	2	13	MDTM	13:53
australian2	13.33	11.63	16.76	**22.54**	1.00	17.17	0.37	5	55	MDTM	2:04
australian3	17.10	11.63	10.98	**15.20**	1.00	11.50	0.31	1	5	MDTM	1:43
balance1	0.00	0.00	0.00	**22.29**	1.00	4.90	8.00	1	7	PPD	0:25
balance2	0.00	0.00	0.00	**22.93**	1.00	8.60	9.99	1	7	PPD	0:35
balance3	0.00	0.00	0.64	**24.20**	1.00	14.33	15.75	1	7	PPD	0:21
cmc1	43.75	45.11	40.92	**23.31**	0.99	44.63	3.21	4	41	MDTM	8:11
cmc2	43.89	38.32	44.99	**18.97**	0.99	45.85	1.83	3	182	MDTM	44:57
cmc3	46.06	45.11	38.21	**21.68**	1.00	42.87	3.22	6	127	MDTM	10:42
concentric1	0.00	0.16	0.00	**23.52**	1.00	2.51	0.99	1	15	PPD	14:29
concentric2	0.80	0.48	1.92	**20.96**	1.00	2.22	0.73	2	92	PPD	25:40
concentric3	0.48	0.80	1.76	**21.28**	1.00	2.00	0.41	2	22	PPD	17:07
diabetes1	25.00	19.27	22.92	**15.10**	1.00	23.23	1.24	2	163	PPD	8:05
diabetes2	22.66	22.40	24.48	**9.90**	0.99	29.11	4.40	1	21	MDTM	3:41
diabetes3	25.00	21.88	24.48	**13.02**	0.99	28.59	6.47	3	23	MDTM	3:23
german1	22.80	24.80	26.80	**9.60**	0.99	29.68	2.73	11	54	ML	11:38
german2	22.60	21.20	27.20	1.60	0.69	28.96	2.02	16	173	ML	13:49
german3	25.00	22.80	28.40	0.80	0.60	27.48	1.53	12	432	ML	27:33
monks1	0.00	0.00	0.00	**27.08**	1.00	7.36	7.63	3	29	ML	0:53
monks2	0.00	0.00	0.93	**24.77**	1.00	7.96	8.77	5	45	ML	1:39
monks3	4.94	9.76	2.78	**16.67**	1.00	2.78	0.00	2	7	MDTM	0:05
satimage1	19.18	19.22	21.68	2.24	0.20	26.17	6.15	18	856	PPD	65:16
satimage2	15.82	14.74	16.46	**7.20**	1.00	19.99	2.96	15	60	MDTM	95:51
satimage3	17.28	16.79	15.34	**4.97**	0.99	17.06	1.82	9	49	MDTM	113:52
segment1	5.89	5.37	4.84	**80.80**	1.00	6.83	1.21	9	28	ML	45:59
segment2	5.97	4.51	4.50	**10.38**	1.00	6.21	1.48	8	92	ML	11:11
segment3	4.24	4.16	6.40	**12.80**	1.00	7.23	0.75	9	111	ML	20:33
smoking1	29.85	30.01	31.19	**32.87**	1.00	52.27	27.00	5	855	PPD	49:07
smoking2	30.13	29.87	30.91	**31.89**	1.00	41.82	21.86	2	393	PPD	10:01
smoking3	30.34	29.87	31.47	**34.41**	1.00	49.66	28.50	2	868	PPD	9:04
spirals1	17.01	22.68	24.22	**24.22**	1.00	28.98	2.76	2	22	ML	9:12
spirals2	26.29	19.59	34.66	**17.12**	1.00	30.69	3.75	2	38	ML	8:34
spirals3	18.56	19.59	32.15	**19.21**	1.00	29.52	4.83	2	101	ML	6:13
titanic1	21.55	20.55	20.15	**12.34**	1.00	20.15	0.01	1	12	MDTM	9:48
titanic2	22.18	22.18	20.15	**11.98**	1.00	19.48	0.49	1	31	MDTM	4:41
titanic3	21.64	22.18	20.69	**11.62**	1.00	21.13	0.68	2	48	MDTM	3:22
vehicle1	9.22	13.27	13.68	-0.94	0.17	13.77	0.86	13	13	ML	11:20
vehicle2	9.93	14.22	17.45	0.94	0.35	18.73	2.13	13	15	ML	45:18
vehicle3	12.29	13.74	16.51	-2.36	0.09	15.71	1.61	12	12	ML	12:53
yeast1	39.62	46.90	46.90	22.10	0.78	57.82	7.93	8	182	PPD	7:19
yeast2	42.72	46.09	47.71	21.02	0.75	59.54	10.93	6	233	PPD	14:01
yeast3	48.25	45.01	52.56	15.90	0.48	53.02	1.36	2	43	ML	6:22

tr. error: the objective function value of the best-of-all-runs individual. Note that this is the percentage error calculated by the RAT algorithm described in Section 5.5.1, and is generally not based on the full training set.

val. error: the validation set percentage error of the best-of-all-runs individual.

test error: the test set percentage error of the best-of-all-runs individual.

Δ error: the improvement in test set percentage error of the best-of-all-runs individual over the test set error of the best classifier on the original data. The best classifier comes from the sub-set of MDTM, PPD and ML used by EPrep for the problem, and has the minimum validation set error from amongst the sub-set.

$1 - \alpha$: 1 minus the McNemar's test confidence level, interpreted as the probability with which the best-of-all-runs individual improved the test set error above that of the best classifier in the set used on the original data.

μ test: the mean test set error over the 10 runs.

σ test: the standard deviation of the sample of test set errors over the 10 runs.

no. ftrs: the number of features contained in the best-of-all-runs individual.

no. nodes: the number of nodes contained in the best-of-all-runs individual.

classifier: the classifier used by the best-of-all-runs individual.

time(h:m): the elapsed time taken for the batch of runs, in hours and minutes.

The parameters used to obtain these results are printed in Appendix D. Note that those values of Δ error that are statistically significant at the 99% level are high-lighted in bold.

The following comments can be made about the results:

- For 36 out of 45 problem domains, EPrep brought about a statistically significant (at the 99% level) improvement in classification performance over the classifiers used.
- The test set error obtained by EPrep was far better than the default error rate for all problems except **german** and **smoking**.
- The 9 data sets for which no significant improvement was obtained over the default error or the error of the original classifier were **german**, **satimage1**, **smoking**, **vehicle** and **yeast**. Of these, **satimage1** is an outlier according to Figure 6.1, **smoking** is suspected to contain no learnable information, and **vehicle** is already well classified by the ML method on the original data. There is no such explanation for the poor performance of EPrep on **german** and **yeast**.
- The features generated usually resulted in good generalisation performance in the sense that the test set error was close to the validation set error.
- The RAT estimate of training error was often similar to the validation set error.
- The time taken to perform 10 runs of EPrep ranged from hours to days.
- The results were fairly consistent for the three permutations of each data set. For **australian** and **satimage**, the test set error was changed dramatically by permuting the data. In 11 of 15 cases, the same classifier was used from the set available for all three permutations. The number of features in the best-of-all-runs individual was

about the same for each permutation of the same data set, while the number of nodes tended to vary quite a bit. Statistically significant improvements were obtained for all three permutations in each case except for **german** and **satimage**.

- In most cases, the mean test set error was fairly close to the best, and the standard deviations were small relative to the error.
- For **german3**, **spirals2**, **spirals3**, **titanic2** and **vehicle3**, the mean test set error was lower than the best. Hence the best run did not correspond to the lowest test set error for these data sets.
- The synthetic problems **balance**, **concentric**, **monks** and **spirals** were not, in general, solved perfectly. While performance was good on **balance**, the best individual did not generalise perfectly for **balance3**. Similarly for **monks**, generalisation was imperfect for **monks2** and the minimum error was not achieved for **monks3**, which contains noise. The results for **spirals** were quite poor, but it should be noted that zero error may be out of the question because the training set is under-sampled.
- For many of the data sets, the dimensionality of the data was significantly reduced by the best pre-processor.
- For **vehicle1** and **vehicle3** the test set error of the classifier on the pre-processed data was worse than on the original data. This occurred because EPrep was not able to improve upon the original input measurements. The reader may be wondering why, if the initial population was seeded with the full set of input measurements, EPrep can do any worse than the original data. The reason is primarily over-fitting: features are evolved that reduce the error specifically on the training data, and happen to perform better than the original measurements on the validation set. When it comes to the test set, however, the over-specialised features do not generalise as well as the original measurements.

Each time EPrep is run on a set of data, it generates many data files, plots and a HTML report summarising the results. The main outputs of EPrep were the following 10 quantities:

best-of-generation fitness: the minimum objective function value (RAT training fitness) of individuals in the population at generation g.

best-of-generation validation error: the validation set error of the best-of-generation individual at generation g.

average fitness: the mean objective function value of individuals in the population at generation g.

standard deviation of fitness: the standard deviation of objective function values of individuals in the population at generation g.

average number of features: the average number of features contained in individuals in the population at generation g.

average number of nodes: the average number of nodes contained in individuals in the population at generation g.

average number of introns: the average number of introns contained in individuals in the population at generation g.

average number of RAT trials: the average number of training samples required to evaluate the fitness of individuals in the population at generation g.

average optimisation improvement: the average improvement in fitness brought about by local optimisation at generation g.

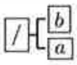
average operator probabilities: the average probability per genetic operator contained in individuals in the population at generation g.

Each quantity was plotted versus generation per run, and averaged over all runs. Therefore the experiments performed resulted in $15 \times 3 \times (10+1) \times 10 = 4{,}950$ plots, which cannot all be included in this thesis. Instead, a single plot per quantity for each problem is included in Appendix E. The plots are included to support the comments on EPrep's performance made in the following sub-sections. Each plot contains the quantity for all of the three permutations, averaged over the 10 runs. The exception is the operator probabilities, since showing three groups of operator probabilities would over-complicate the plot. Instead, only the probabilities for the first permutation are shown. The specific plots for each run can be viewed through the HTML reports using a web-browser. All HTML reports generated in these experiments are found on the CD-rom included with this thesis; instructions for the use of the CD-rom are found in Appendix A.

Since the plots in Appendix E are averages, they do not capture the dynamics of individual runs. Therefore the reader should be careful in interpreting the dynamics of the averaged plots, since they are generally not reflected in the curves from the individual runs. In the following sub-sections, some examples of the evolved pre-processors are presented, and characteristics of the plotted quantities are discussed.

6.4.1 Pre-Processors

In this section, a selection of the pre-processors evolved during the experiments is shown. Each pre-processor is displayed as a set of trees, each tree being a feature. The trees are displayed horizontally, and the order of the function arguments is from bottom to top; *ie:* the bottom-most argument is the first. For example, the tree:



represents the expression a/b.

The pre-processors shown here do not constitute the full results of the experiments, but are representative of the results obtained. In each case, the pre-processor shown is the best-of-run individual from the best run on the corresponding permutation of the data set. Therefore the pre-processor labeled **australian1**, say, corresponds to the **autralian1** entry in Table 6.4. Note that some of the pre-processors cannot be shown here because they are too large to fit on a page, and would not make much sense to the reader anyway.

The pre-processors evolved for the synthetic problems are shown in Figure 6.2. These results are easier to analyse, because the true distributions of the classes are known. Referring to Appendix C, the true solution to the **balance** problem is based on the difference between X1×X2 and X3×X4. Since the single feature in Figure 6.2(a) reduces to (X1.X2)/(X3.X4), EPrep found the perfect solution. The **concentric** data are dichotomised by the circular boundary between the two clusters; see the diagram in Figure C.1. The boundary is defined by the circle centred at (0.5, 0.5):

$$\{(x, y) : (x - 0.5)^2 + (y - 0.5)^2 = 0.3^2\}$$

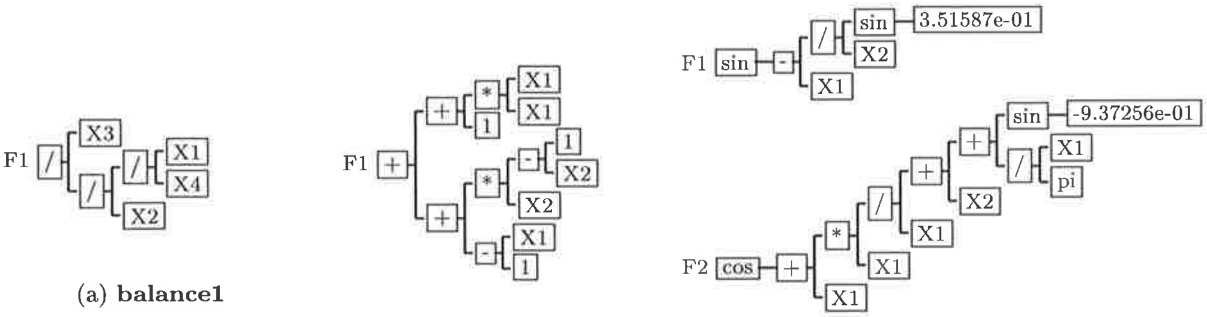
The expression simplifies to:

$$x^2 + y^2 - x - y = -0.41 \tag{6.1}$$

The feature evolved by EPrep, shown in Figure 6.2(b), reduces to:

$$X1^2 + X2^2 - X1 - X2 + 1$$

With the appropriate substitution of X1 for x and X2 for y, this is identical to Equation 6.1 except for the constant factor. It is the constant factor that is effectively modified by the classifier, which acts like a fine-tuning mechanism. The validation set error is not zero in Table 6.4 due to finite data effects. That is, the value of -0.41 is not perfectly obtained from the finite training set, so some of the boundary points are misclassified on generalisation.



(a) **balance1**

(b) **concentric1**

(c) **spirals1**

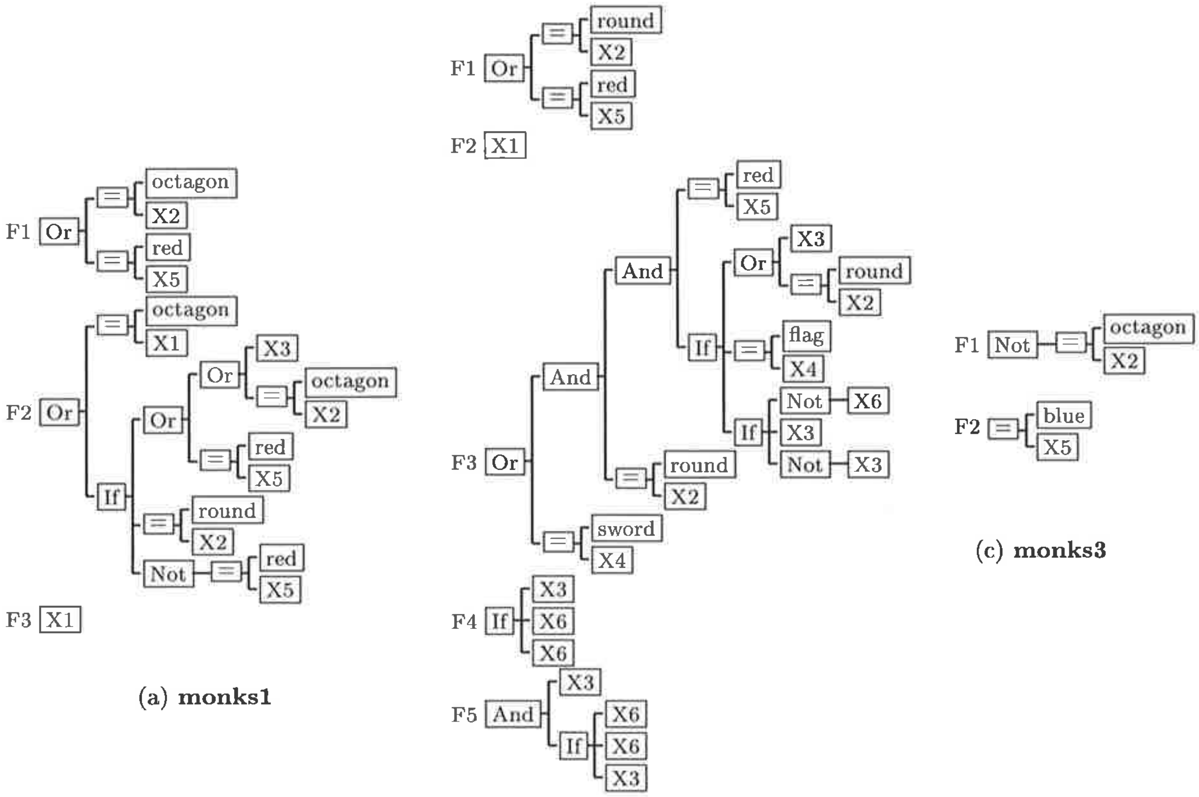
Figure 6.2: Best feature sets evolved by EPrep for synthetic problems.

EPrep had some trouble classifying the **spirals** data, obtaining a best test set error of 24.22% on **spirals1**.

The best pre-processors for the three **monks** problems are shown in Figure 6.3. As is usual for the results of genetic programming, we find some redundant expressions that can be simplified by hand, and others that do not contribute to the fitness of the solution. For example, in Figure 6.3(b), F5 contains "If(X3 X6 X6)", which simplifies to X6.

For **monks1**, the concept to be learned was (head_shape(X1) = body_shape(X2) OR jacket_colour(X5) = red) (refer to Appendix C). EPrep learned this in some way, since the test set error was zero. However, the results are not as neat and nice as we would have hoped. From Figure 6.3(a) we see that EPrep has discovered =(X5 red), but the variables X1 and X2 are not considered to be of equal type since they are enumerated variables, which in general do not possess the same enumerated values. Therefore EPrep cannot generate the expression =(X1 X2); it circumvented this restriction by increasing the dimensionality and using the separating qualities of the classifier. Since X1 and X2 can take on the values round, square and octagon, these were tested separately in different features of the pre-processor so that the conjunction of X1 and X2 having some particular value is transformed to a corner of the output-domain hyper-cube. These corners can then be separated by the ML classifier, which has the advantage that diagonally-opposite corners can be included in the same class. The solution also contains the superfluous variable X3, which has "hitch-hiked" along with the best solution.

For **monks2**, the concept to learn is whether exactly two of a robot's six attributes have their first value. EPrep almost learned this concept, making four errors on the test set. For X1 and X2, the first value is round; for X4 sword and X5 red. X3 and X6 are boolean, so the first value is false. We can see these first values being used in the solution of Figure 6.3(b). Trial runs without the ML classifier have shown that EPrep is unable to discover the true solution without the ML classifier, whose special properties are the key to discovering the classification rule. The problem is that, when classifying the training and validation samples, the classifier is only trained on the training set, whereas to classify the test set the union of the training and validation sets is used for training. The features evolved are not robust to



(a) **monks1**

(b) **monks2**

(c) **monks3**

Figure 6.3: Best feature sets evolved by EPrep for the **monks** problems.

this variation in training data, which overall stems from the very small data set available, and 4 of the test samples are misclassified.

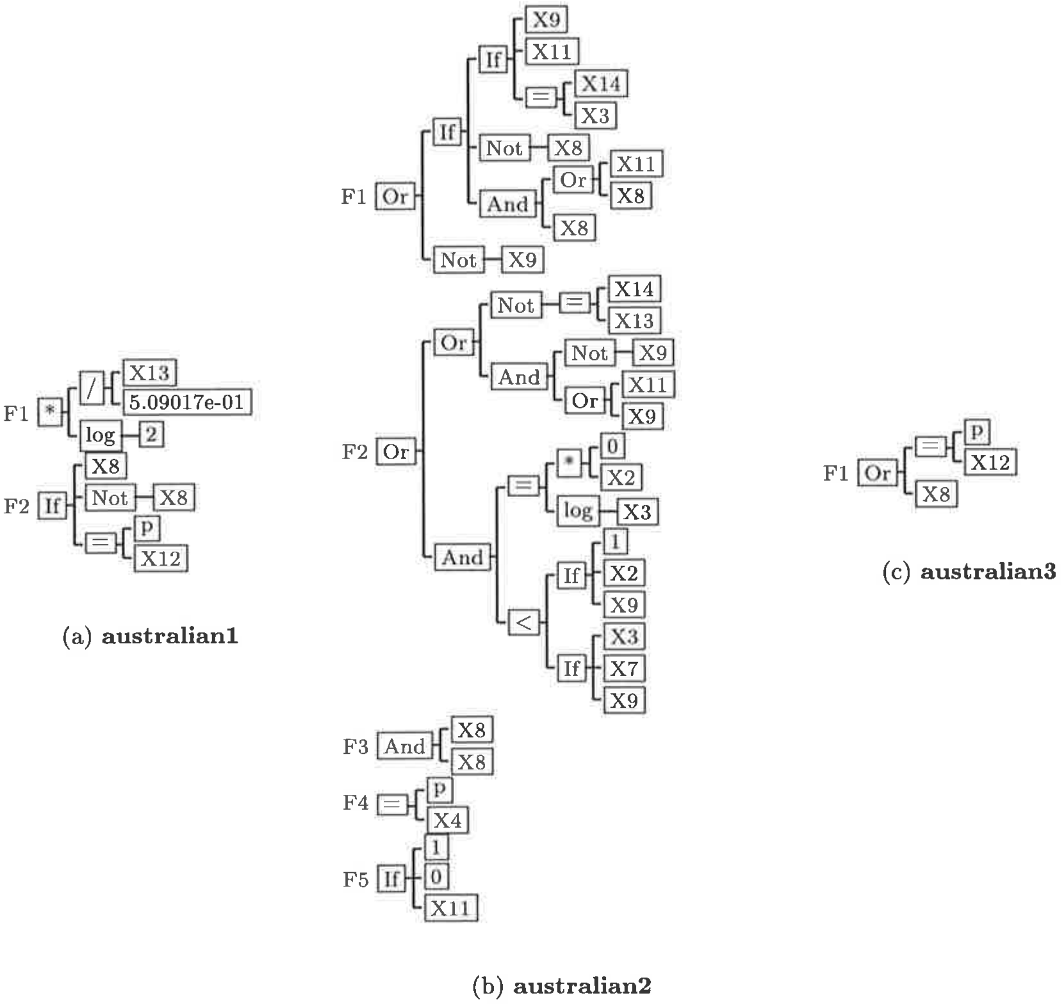
In the third **monks** problem, the concept is (jacket_colour(X5) = green AND holding(X4) = sword) OR (jacket_colour(X5) != blue AND body_shape(X2) != octagon), where != means "not equal to". Although EPrep managed to learn the second AND term in the concept, it was unable to detect the first term. Indeed, X4 does not even feature in the solution. A suitable explanation can be found by closer examination of the data, which is graphically presented in (Thrun *et al.*, 1991). Of the 122 training samples, there are only two samples from class 1 that are covered by the first term and not by the second. Further, both of these examples are found in the training set, so when choosing the best run based on validation set error, an individual that has learned the first part of the concept is indistinguishable from one that has not. Too little evidence is provided by which EPrep can learn the first term rather than the added noise. In short, the training and validation sets are not sufficiently representative of the whole data set to properly learn the first part of the concept. EPrep made 12 errors on the test set; these are the 12 samples that are covered by the second term but not the first.

The remaining pre-processors shown here are for the real-world data sets. Figure 6.4 shows the best features evolved for each of the three permutations of **australian**. The GPPs for **australian1** and **australian3** are quite similar, with X8 and X12=p being the common expressions. For **australian2**, however, the GPP is much larger, and does not contain the variable X12 at all. This shows just how much the evolved features can vary due to a permutation of the data. It is interesting to note that for **australian3**, only X8 and X12 were needed, whereas the documentation of the data set states that a step-wise regression procedure strongly suggested that attributes X5, X8, X9, X13 and X14 were the relevant measurements. The latter result is, however, only step-wise optimal.

The best-of-run GPPs vary not only with the permutations of the data, but also between runs on the same permutation. For example, compare the GPP from the best run on **australian3**, shown in Figure 6.4(c), with training-validation-test set errors 17.10-11.63-10.98%, against the BOR pre-processor from another run shown in Figure 6.5 with errors 15.94-11.63-11.56%. Although the two pre-processors have the same validation set error, the second is much larger and uses a larger set of input variables. The training set error of the larger individual is lower than that of the smaller because it has started to over-fit the data, but the test set error is higher because it does not generalise as well.

The best feature set for **titanic1** and **titanic2** are shown in Figure 6.6. The single feature for **titanic1** in Figure 6.6(a) translates to "a passenger survived if he/she was a child or female, and not in third class". This is the familiar women-and-children-first rationale, with discrimination against third class. Slightly different results were obtained for **titanic2**. The feature in Figure 6.6(b) simplifies to (child AND 3rd class AND male) OR (adult AND (male OR 3rd class)). In words, "a passenger did not survive if he/she was a 3rd class male child, an adult male, or an adult from third class". The difference from the first permutation is that little girls from third class survived. This difference arose due to the permutation of the data, so the whole data set would have to be used to ascertain the true facts.

In Figures 6.7 and 6.8, pre-processors for some of the other real-world data sets are shown. The GPP for **cmc1** in Figure 6.7(a) does not use X5 (religion: islam or non-islam?) or X6 (wife now working?). For **diabetes3** in Figure 6.7(b), the attributes X1 (number of times pregnant), X4 (triceps skin fold thickness) and X5 (2-hour serum insulin) are not needed for classification of the data. For the **segment1** problem, the solution shown in Figure 6.7(c) only uses 6 of the original 11 input measurements. These results indicate that EPrep performs *attribute selection* at the same time as performing feature extraction. The information about which attributes are useful for classification must be handled with care, though, because some attributes may be present but not contribute to the fitness of the



(a) **australian1**

(b) **australian2**

(c) **australian3**

Figure 6.4: Best feature sets evolved by EPrep for **australian** problem.

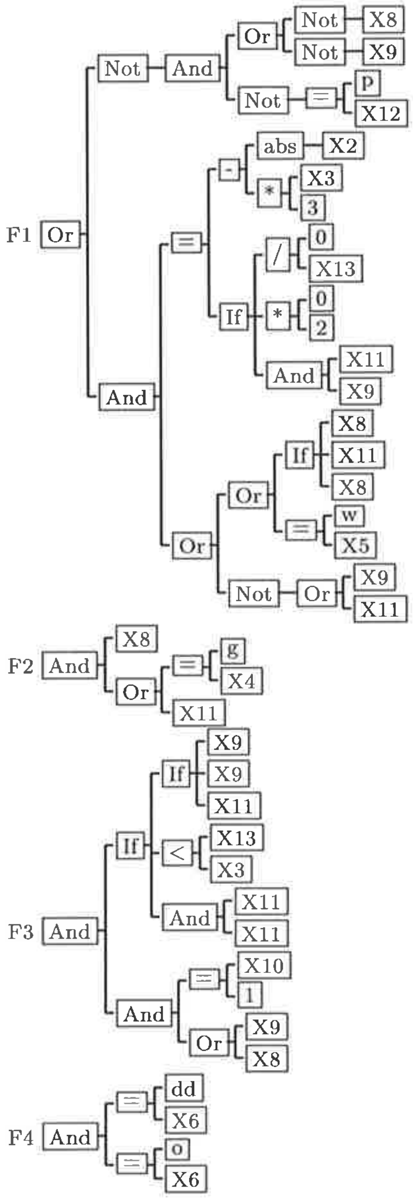
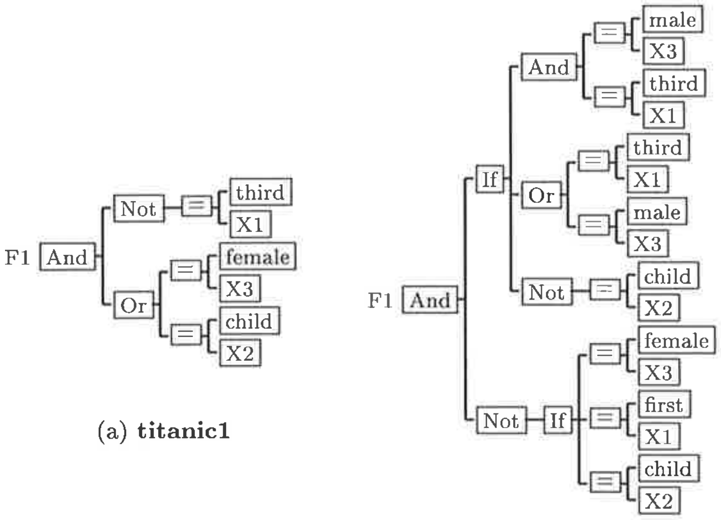


Figure 6.5: Best feature set from run 9 for **australian3**.



(a) **titanic1**

(b) **titanic2**

Figure 6.6: Best feature sets evolved for first two permutations of **titanic**.

individual, and the results vary from run to run.

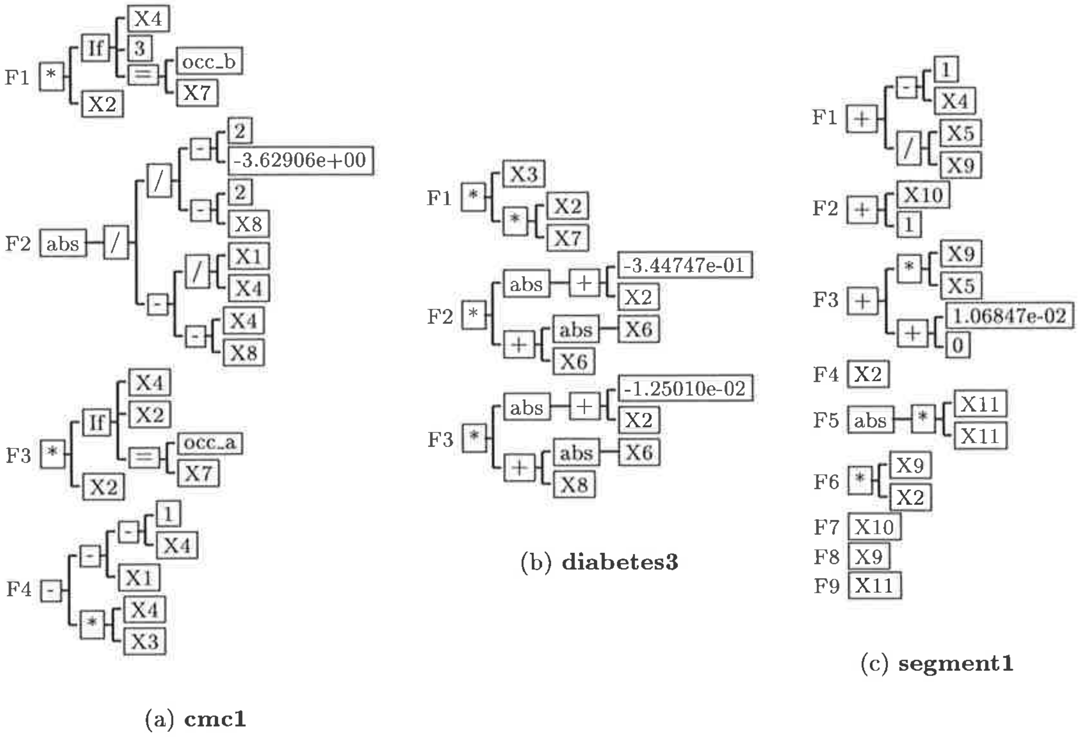
The GPP for **vehicle2** consists of virtually un-modified attributes. It would appear that the ML classifier is already so effective that transforming the attributes could not improve the classification rate, so EPrep reverted to feature selection, finding the most economical combination of the original attributes through high-level crossover. The solution shown here uses only 12 of the original 18 attributes. The pre-processors for **abalone1** and **satimage2** are quite large, and have departed the limits of what can reasonably be interpreted by a human. The solution for **abalone1** does not use X1, which is the sex of the mollusk.

6.4.2 Evolution of Fitness

The figures in Appendix E show the best-of-generation, average and standard deviation of fitness, along with the best-of-generation (BOG) validation error for each data set as a function of generation. Note that fitness in this case is a classification error, so lower is better.

The reproduction of the top N_{rep} individuals in the population ensures that the BOG individual is propagated to the next generation. Therefore it may surprise the reader that the BOG fitness and validation set error is not a strictly monotonic non-increasing function of the generation. The reason is that the fitness criterion is changing each generation because some of the samples are sorted to the start of the training set. Difficult samples are moved to the front of the list, and in some cases the lowest error in the population is no longer as low as that from the previous generation. This changes the criterion for selecting the BOG individual, so the validation set error can increase.

The BOG fitness, based on training set error, generally decreased with time, with the exception of the **vehicle** problem. For some of the data sets (*eg:* **australian**, **diabetes**, **satimage**), there was a characteristic bump in the plot as the fitness initially increased, then decreased with generations. This is to be expected: at generation 0, there are some individuals in the population that have a low objective function value but do not generalise well, or *specialists*, and some with low fitness that do generalise relatively well, or *generalists*.



(a) **cmc1**

(b) **diabetes3**

(c) **segment1**

Figure 6.7: Best feature sets evolved by EPrep for real-world problems.

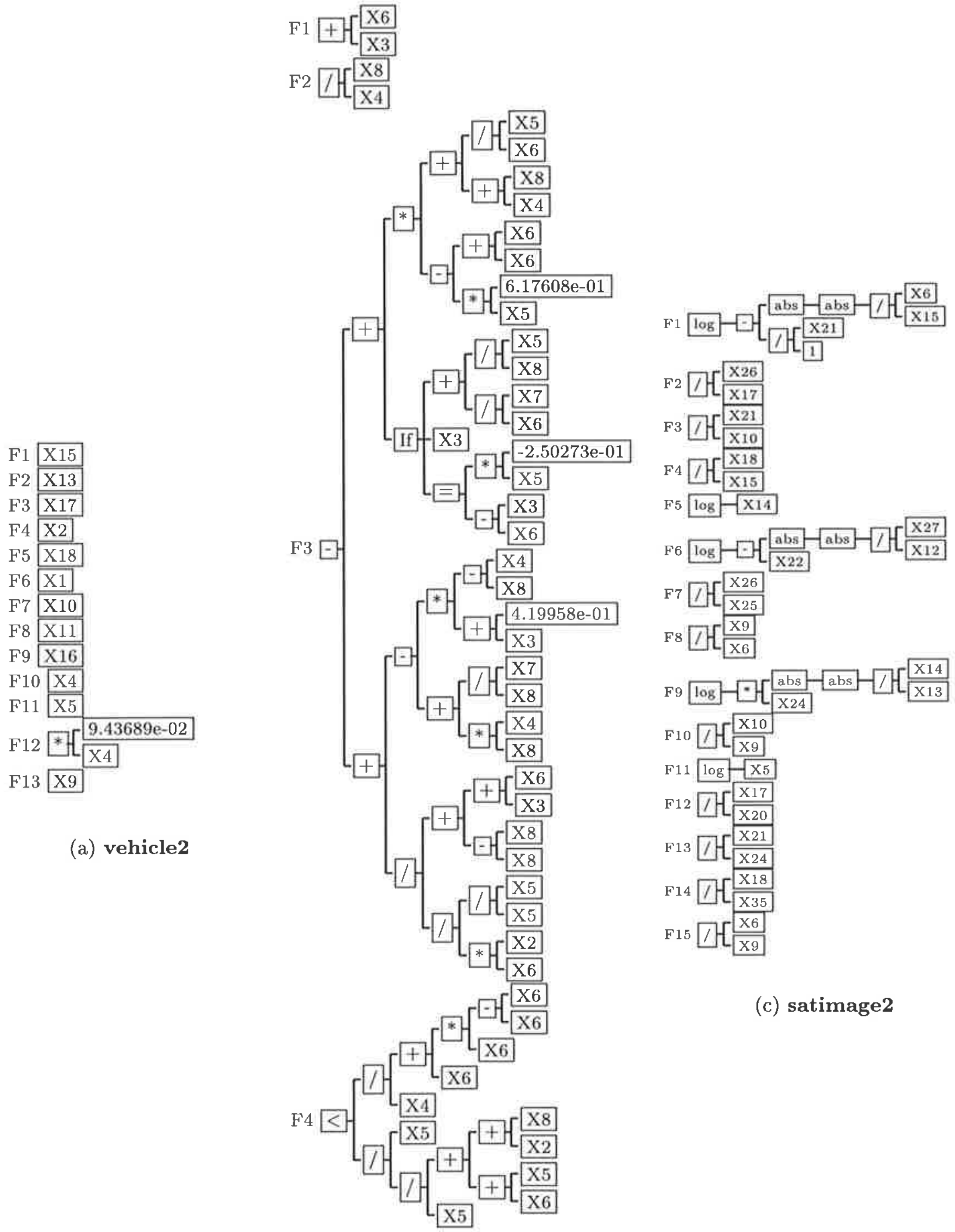
On the first generation, there is no way to tell the difference between a specialist and a generalist. Over the first few generations, the specialists are gradually removed from the upper ranks of the population because their fitness does not remain low as the objective function changes. The less brittle generalists remain around the top of the ranks, and gradually learn to classify the new difficult samples through the genetic operators' modifications. After several generations, the gross structure of the data has been learned, and the error starts to decrease.

In most cases, the plots of BOG fitness were similar for the three permutations of the data. When the plots were different, they usually had roughly the same shape but differed by a bias; *eg:* **australian**, **diabetes**, **titanic**, **satimage**. The implication is that some permutations of the data deprive the training set of samples that are essential to approach the Bayes rate. In particular, the first permutation of **satimage** was significantly more difficult for EPrep than the other two permutations.

In the BOG validation set error, there is a characteristic decrease to a minimum value, and then an increase as over-fitting occurs and generalisation capabilities are lost. As an example, the best-of-generation validation set error is plotted in Figure 6.9 for the best run of **segment3**. The validation set error was fairly volatile in general, except for the **monks** and **concentric** data sets.

Average fitness generally decreased with time, and shared the bump characteristic with BOG fitness early in the run. The standard deviation of fitness increased most of the time, but sometimes decreased. In both cases, it often stabilised around some value in the steady-state, indicating that the genetic operators were providing a good degree of genetic diversity. The behaviour of the fitness variance is problem-dependent, and relies on the amount of reproduction and tournament size as well.

In most cases, EPrep converged quickly upon the solution when compared with typical run durations for GP. As an extreme example, the best solution for **australian3** was found



(a) **vehicle2**

(b) **abalone1**

(c) **satimage2**

Figure 6.8: Best feature sets evolved by EPrep for real-world problems.

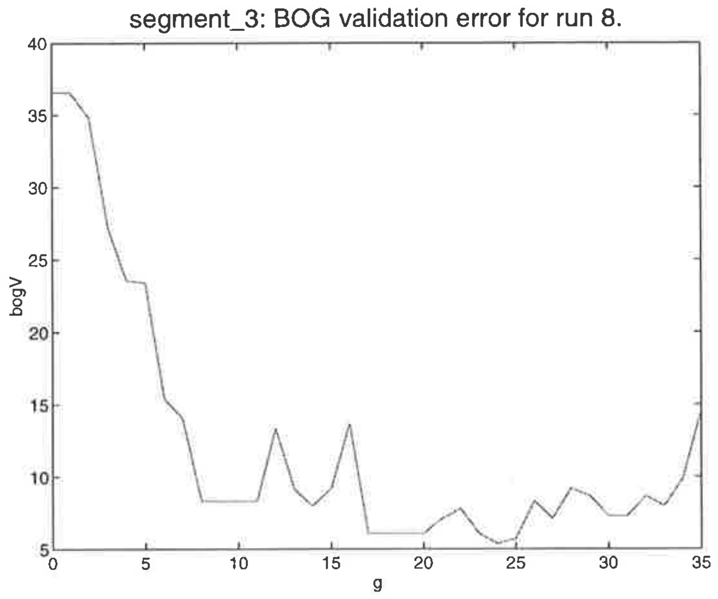


Figure 6.9: Plot of best-of-generation validation set error (%) versus generation for the best run of **segment3**.

in the first generation, as shown by the plot of validation set error versus generation in Figure 6.10. On the other hand, **cmc2** and the **yeast** problem took 200 and 150 generations to complete respectively.

6.4.3 Evolution of Size

The figures in Appendix E show the average number of features and nodes per individual plotted versus generation for each data set. Apart from **australian**, **titanic** and **yeast**, the average number of features increased as a function of generation. This indicates that the use of more features is advantageous for separating the data. For **german**, **segment** and **vehicle** there was a characteristic trough in the number of features after several generations. In those cases, it may be that the optimisation process focused on a particularly valuable, small set of features and later added to their number to improve accuracy. Note that the trough is much more pronounced for **german2** than for **german1** and **german3**. Presumably this is a consequence of the different parameters used for **german2**. Since several parameters were changed for this permutation, the trough cannot be attributed to any single parameter.

The behaviour of the average number of nodes per individual was problem-dependent. In some cases the number of nodes gradually decreased, in others it increased and for some it stayed the same in the steady-state. In those cases where the number of nodes continued to increase, it was a slight increase that ended in a roughly constant level; *eg:* **segment** and **balance2**. Therefore bloating did not occur in the same sense that it traditionally does in GP: the BOG fitness was still changing with a steady increase in program size, indicating that more and more nodes were being added to the pre-processors to improve their accuracy on the training set. Eventually this would lead to over-fitting.

6.4.4 Evolution of Operators

In Appendix E, the average number of introns per individual and the average operator probability per individual for each genetic operators is shown versus generation for each data set. The number of introns in the population generally fell or stayed the same as

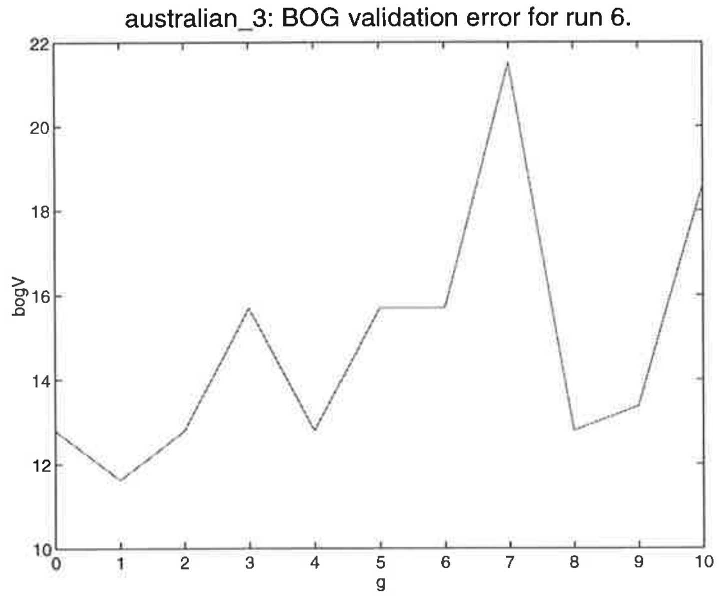


Figure 6.10: Plot of best-of-generation validation set error (%) versus generation for the best run of **australian3**.

evolution proceeded. This indicates that the evolutionary algorithm was not utilising the introns to protect highly-fit schemata as predicted. There are two implementation details that may have interfered with the accumulation of introns. First, the high-level crossover operator was biased towards a decrease in the number of introns in the offspring, the reason being that in special cases, such as one of the parents having only a single feature, introns left hanging off the ends of the offspring were discarded. Second, a closer examination has found that the delete-feature mutation operator had a bug in it: the deleted feature was randomly selected from amongst the T features present, rather than selecting a point in the whole chromosome including the features and introns. Consequently introns could not protect features from delete-feature mutation. Nevertheless, it was expected that the number of introns would increase dramatically with generations in a similar fashion to the bloating phenomenon through the operation of high-level crossover.

For 34 of the 45 data permutations, high-level crossover had the highest probability averaged over runs, generations and individuals. On the surface, this suggests that high-level crossover is useful in almost any situation. However, there was a natural bias towards the high-level crossover operator, because both offspring from a crossover inherited the operator probabilities from the parent used to select the operator. Therefore the ascendancy of this operator may be purely due to the bias. Apart from that, there was generally no pattern to the operator probabilities, and the plots from individual runs deviated significantly from the averaged plot.

6.4.5 Evolution of Training

In Appendix E, the average number of training samples examined per individual and the average improvement in fitness due to local optimisation are shown versus generation for each data set. The number of RAT trials generally increased with generations, ending at some number significantly less than n_{tr}. This behaviour was anticipated, since fewer samples should be required to distinguish individuals in the initial random population, but more and more samples should be required to distinguish similar solutions that appear as the population converges. To illustrate, the settings for **abalone2** were $n_{tr} = 2088$ and

$N_o = 750$. From Figure E.1, the average number of trials needed per individual at generation 0 was approximately 880, and after 15 generations the number of trials had risen to about 1150, which is still around half of the full training set.

The number of RAT trials was constant for the **monks** and **spirals** problems, since the training set was so small that all the samples were used to evaluate fitness. For **german** and **vehicle**, there was a characteristic dip in the number of trials after several generations. In the case of **german**, this dip coincides with the dip in the average number of features per individual. The change in models could be related to the reduced number of trials, because classifiers using fewer features may be easier in general to distinguish. For example, suppose that two offspring are generated from a parent individual, both via the mutation of the first feature F1. The first offspring a_1 has a new feature F1 with improved discriminatory properties, while a_2 has an F1 whose discriminatory powers have been destroyed through the mutation. If there are many features in each individual, then the classifier may be able to classify the data in the sub-space of a_2's features that do not include F1 fairly well, and more samples will be required to distinguish the performances of a_1 and a_2. If, on the other hand, there are few features in the individual, then the new feature plays a more important role, and the poor F1 in a_2 cannot be disguised as effectively by the other features. Therefore fewer trials are necessary to distinguish the individuals.

There was little consistency between the optimisation improvement plots for the different data sets. In some instances, optimisation was more effective at the beginning of the run and quickly tapered off, while in other instances its effectiveness increased with generations. The average improvement never rose above 5.3% error for any problem, and was most often less than 1%. There was no improvement for **monks3** and **yeast** because optimisation was not used for these problems. The usefulness of optimisation is questionable, since it increases the execution time of EPrep by a multiplicative factor.

6.4.6 Other Observations

The HTML reports generated by EPrep and included on the CD-rom contain extra information about the simulations. This sub-section contains salient points from those sources of information.

The confusion matrices (explained in Section 2.4.3) of the best-of-all-runs individuals were examined. The confusion behaviour for **abalone** was interesting in that classes were generally confused with numerically-adjacent classes. The confusion matrix for **abalone2** is shown in Table 6.5. For example, of the samples belonging to the seventh class for **abalone2**, 28 samples were classified as belonging to the sixth class, 28 were correctly assigned to the seventh class and 30 samples were misclassified as belonging to the eighth class. This behaviour is understandable because the class labels correspond to age and have a natural ordering. That is, we are more likely to misclassify a 10-year-old abalone as an 11-year-old than as a 20-year-old.

For the **diabetes** problem, EPrep tended to mistake a large portion of the diabetics for non-diabetics. Similarly for **german**, people with bad credit tended to be mistaken for people with good credit. In the **titanic** domain, the classifier was biased towards non-survivors. In all three of these cases, the class to which EPrep was biased had a larger number of samples in the test set than the other class.

In **satimage**, there was confusion between grey (third class), damp grey (fourth class) and very damp grey (sixth class) land types. This confusion is understandable. In the **vehicle** problem, the main confusion was between the first and second classes, the Opel and the Saab. Again this is understandable, since both vehicles are sedans and the other two classes are bus and van. The confusion matrices for the **smoking** problem verified that EPrep was guessing the most frequent class, the third class.

Table 6.5: Confusion matrix for **abalone2**.

Class		Predicted																												Total
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	
Actual	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
	3	1	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**5**
	4	1	0	3	9	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**15**
	5	1	0	3	6	8	10	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**29**
	6	4	0	0	0	5	25	21	8	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**65**
	7	3	0	1	0	2	28	28	30	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**97**
	8	0	0	0	0	0	10	20	58	23	11	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**135**
	9	6	0	0	0	0	1	13	37	44	27	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**168**
	10	7	0	0	0	0	5	8	15	28	23	70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**157**
	11	4	0	0	0	0	1	5	11	13	14	73	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	**122**
	12	2	0	0	0	0	0	3	6	9	5	41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	**68**
	13	10	0	0	0	0	2	0	6	3	5	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	**51**
	14	3	0	0	0	0	0	1	0	6	3	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**32**
	15	5	0	0	0	0	0	0	2	4	0	14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	**26**
	16	1	0	0	0	0	0	0	0	3	3	8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	**18**
	17	0	0	0	0	0	0	0	0	3	1	7	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	2	0	0	**15**
	18	0	0	0	0	0	0	0	2	0	1	7	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	**11**
	19	1	0	0	0	0	0	0	0	0	2	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	**8**
	20	0	0	0	0	0	0	0	0	1	0	4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	**7**
	21	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**4**
	22	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**
	23	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**3**
	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**1**
	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**1**
	26	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	**1**
	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
	29	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
Total		**53**	**0**	**11**	**16**	**16**	**82**	**101**	**175**	**140**	**99**	**327**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**7**	**0**	**0**	**17**	**0**	**0**	**1045**

The average number of fitness evaluations during a run was problem-dependent. The largest was for **cmc2**, which used over 886,000 fitness evaluations per run.

Most of the runs were terminated by the generalisation loss criterion or by reaching the maximum number of generations. Zero validation set error was sometimes the reason for stopping for the synthetic data sets, and the training progress criterion occasionally fired.

Generalisation was generally good, in that the best run had a test set error that was the lowest or very close to the lowest of the ten runs. For 9 of the data sets, however, the test set error of the best run was significantly higher than the lowest. These data sets were **cmc2**, **diabetes1** and **3**, **german3**, **spirals2** and **3**, **titanic2**, and **vehicle2** and **3**. The poor generalisation for **spirals** indicates that EPrep was badly over-fitting the training data. The below-average generalisation for **diabetes** is attributable to the relatively small number of samples in that data set.

The pre-processors themselves varied significantly for the different runs by the number of features, nodes and distinct input variables.

6.5 Experimental Results of MLP

The results of the architecture-search runs are shown in Table 6.6. For each data set, the pivot architecture is shown, along with its validation set error, number of weights and the number of epochs required to train it. These pivot architectures were then run 10 times to obtain the results shown in Table 6.7. This table contains the mean and standard deviation of training, validation and test set error and number of training epochs for each of the data sets over the 10 runs, as well as the test set error of the best run and the time in hours and minutes for the architecture search and the ten runs combined.

From the results, the following general observations can be made:

- The classification errors obtained by the MLP are lower than the default error rate by a significant margin for all data sets except **spirals** and **smoking**.
- In most cases, the pivot architecture was different for different permutations of the data.
- Execution times ranged from minutes to days.
- Classification error rates were fairly similar from permutation to permutation.
- The mean test set error was fairly indicative of the best error except for the cases of **balance2**, **balance3** and **monks2**, which had a much lower best test error.
- **vehicle2**, **vehicle3**, **yeast1**, **yeast3**, **german3**, **diabetes1** and **balance1** had a higher test set error from the best run than the average. In these cases, the minimum validation set error did not correspond to the minimum test set error.

6.6 Experimental Results of Decision Trees

The results of the QUEST experiments are shown in Table 6.8. For each permutation of each data set, the 10-fold cross-validation classification error, test set error and number of leaf nodes in the tree are shown. The number of leaf nodes in the tree can be interpreted as the number of regions into which the feature space is divided. The execution time is not listed, but times ranged from a few seconds to 8 minutes.

From the results, the following general observations can be made:

- The results are fairly robust to permutation of the data.
- It is common for the CV error to be larger than the test set error. The reason for this is the CV estimates of error are based on fewer training samples than the test set error.
- The trees for **abalone**, **german3** and **smoking** contain fewer leaf nodes, and therefore fewer decision regions, than classes.
- All of the test set errors are lower than the default error rate for the data set, except for **smoking** and **german3** for which the tree has a single node and guesses the most frequent class.

Some of the decision trees obtained are shown in Figures 6.11, 6.12 and 6.14-6.17. Note that in most cases, the nodes near the root of the tree are identical for the different permutations of the data; that is, the same variables are usually chosen first for splitting. In Figure 6.11, the tree obtained for all three permutations of the **australian** data set is shown. The tree splits on only one attribute, X8. In the diagram, split nodes are circular and leaf nodes are square. The number in the centre of each node is a unique label given to that node. On the left of each split node is the criterion for splitting. If this criterion is true, the decision proceeds down the left branch, otherwise down the right branch. The number to the left of each leaf node is the number of training samples assigned to that node. The number under each leaf node is the class to which samples falling in that node are assigned.

The decision tree for **concentric1** is shown in Figure 6.12. The decision regions created by this tree are shown in Figure 6.13. We can see that the decision tree approximated the circular shape with a piece-wise linear boundary. The resolution of this boundary was determined by the amount of training data available.

Table 6.6: Architecture search results for MLP.

Problem	architecture	best val. error %	no. weights	no. epochs
abalone1	32+0*l*	71.552	1309	250
abalone2	32+0*l*	73.084	1309	300
abalone3	32+0*l*	73.563	1309	250
australian1	04+2*l*	11.047	180	1800
australian2	24+0*s*	9.884	1034	700
australian3	04+4*s*	12.791	194	1650
balance1	04+2*l*	0.641	39	3000
balance2	08+8*l*	1.923	139	3000
balance3	16+8*l*	1.923	243	2400
cmc1	16+8*s*	42.935	371	450
cmc2	16+0*l*	41.848	259	550
cmc3	24+0*l*	41.033	387	500
concentric1	16+8*l*	0.320	202	3000
concentric2	16+8*s*	0.640	202	3000
concentric3	16+8*l*	0.480	202	3000
diabetes1	32+0*l*	20.833	354	600
diabetes2	24+0*l*	21.354	266	700
diabetes3	16+0*s*	19.792	178	450
german1	16+8*s*	24.000	1178	2300
german2	04+2*s*	21.200	272	2100
german3	32+0*s*	20.000	2114	1300
monks1	32+0*l*	0.000	578	3000
monks2	08+8*l*	0.000	218	3000
monks3	08+0*l*	7.317	146	3000
satimage1	16+8*l*	10.752	782	3000
satimage2	32+0*l*	9.820	1382	3000
satimage3	16+8*l*	10.503	782	2450
segment1	16+8*l*	3.466	391	2600
segment2	16+8*l*	2.773	391	3000
segment3	32+0*l*	4.159	615	3000
smoking1	32+0*s*	29.692	835	450
smoking2	24+0*l*	29.832	627	700
smoking3	32+0*s*	29.972	835	450
spirals1	16+8*l*	31.959	202	2950
spirals2	16+8*s*	29.897	202	1000
spirals3	16+8*l*	34.021	202	1400
titanic1	32+0*l*	20.545	354	100
titanic2	32+0*l*	22.182	354	100
titanic3	32+0*l*	22.545	354	100
vehicle1	32+0*s*	18.009	740	2800
vehicle2	32+0*l*	15.166	740	1900
vehicle3	32+0*s*	15.166	740	3000
yeast1	32+0*l*	38.814	618	800
yeast2	32+0*l*	40.701	618	900
yeast3	32+0*l*	35.040	618	750

Table 6.7: Classification results for best MLP architectures.

Problem	training %		validation %		test %		best	epochs		time
	μ	σ	μ	σ	μ	σ	test %	μ	σ	(h:m)
abalone1	75.584	0.599	76.925	1.757	75.809	1.605	73.301	35.00	8.16	2:02
abalone2	74.607	0.939	76.341	1.057	77.321	1.100	75.407	30.50	9.26	2:00
abalone3	75.038	1.251	76.360	1.021	76.172	1.482	75.311	29.50	10.12	1:01
australian1	7.623	1.176	13.372	0.613	13.468	0.820	13.873	51.50	14.92	1:01
australian2	7.333	2.041	10.756	0.919	18.208	0.734	17.341	118.00	45.17	1:01
australian3	8.928	0.935	14.593	0.748	11.272	0.734	10.983	66.50	19.59	0:00
balance1	6.186	2.099	10.962	1.108	7.134	1.964	7.643	155.00	44.78	4:03
balance2	5.962	2.237	8.205	2.488	7.643	2.383	1.274	143.50	123.13	4:00
balance3	3.974	2.635	6.282	1.954	7.389	2.788	2.548	192.50	178.39	0:00
cmc1	38.098	2.482	46.658	1.537	43.117	2.074	40.650	111.00	20.52	4:04
cmc2	40.435	1.896	43.261	1.305	46.748	1.926	45.257	72.50	17.99	4:01
cmc3	42.079	3.920	43.859	1.256	41.463	2.649	39.024	90.50	37.23	1:00
concentric1	0.688	0.417	0.976	0.409	2.192	0.908	1.120	2780.00	695.70	13:56
concentric2	1.104	0.594	1.904	0.725	2.192	0.735	1.440	3000.00	0.00	14:44
concentric3	0.384	0.145	1.344	0.560	1.712	0.512	1.280	3000.00	0.00	15:31
diabetes1	16.094	1.537	23.490	0.794	22.135	0.859	22.917	86.50	19.87	2:02
diabetes2	19.792	1.705	23.906	1.055	23.906	1.355	22.396	50.00	38.87	2:00
diabetes3	20.000	2.086	19.427	0.852	21.094	0.563	20.833	120.00	29.25	0:00
german1	15.520	2.821	27.760	1.878	26.920	1.969	25.600	26.50	7.47	0:00
german2	18.100	7.287	27.240	1.899	26.400	2.199	24.000	54.00	33.32	0:02
german3	16.100	1.905	20.360	0.479	20.960	1.375	22.400	51.50	8.83	2:02
monks1	0.000	0.000	0.233	0.735	0.371	0.897	0.000	1014.00	587.28	1:01
monks2	20.000	15.549	25.536	17.759	22.824	15.807	0.000	212.00	347.09	1:10
monks3	1.728	1.193	9.756	0.000	0.972	0.754	0.463	75.00	46.49	13:05
satimage1	9.456	0.998	13.400	0.767	14.413	0.427	13.797	238.00	51.60	53:15
satimage2	12.191	0.469	13.468	0.562	13.250	0.586	12.989	165.50	22.54	22:29
satimage3	10.398	1.186	13.605	0.591	12.293	0.791	11.187	225.00	34.08	29:34
segment1	3.931	0.762	6.014	1.132	6.263	1.661	5.190	465.50	148.78	3:57
segment2	4.597	1.290	4.662	1.116	5.294	1.887	3.460	449.50	89.80	59:15
segment3	5.177	0.806	7.574	0.712	8.495	0.821	7.612	323.00	79.31	48:53
smoking1	30.112	0.932	30.828	0.934	30.672	0.093	30.672	344.17	256.13	33:44
smoking2	29.516	1.215	30.902	0.850	30.938	0.154	30.812	345.83	254.21	39:17
smoking3	30.203	0.087	30.921	0.829	30.882	0.151	31.092	343.06	257.43	9:01
spirals1	44.819	0.560	46.804	4.512	48.979	2.066	45.833	21.50	2.42	13:14
spirals2	43.420	1.966	48.866	2.440	49.125	2.409	50.417	24.50	14.23	50:01
spirals3	40.207	3.413	47.113	3.506	48.271	2.248	46.250	28.50	4.74	33:57
titanic1	21.455	0.000	20.727	0.000	20.690	0.000	20.690	46.00	3.94	28:02
titanic2	21.036	0.115	22.545	0.148	20.036	0.454	20.145	46.00	5.68	6:25
titanic3	19.909	0.000	22.545	0.000	21.597	0.000	21.597	45.50	4.38	19:19
vehicle1	16.147	0.997	25.308	1.207	21.651	1.548	19.340	191.00	23.66	32:36
vehicle2	9.196	2.733	18.199	0.953	21.745	1.482	22.642	324.00	95.77	32:21
vehicle3	17.116	0.642	24.787	1.073	17.830	2.666	18.868	170.00	25.82	22:22
yeast1	38.423	0.899	40.485	0.694	40.296	1.166	41.240	110.50	15.17	11:17
yeast2	37.439	0.612	44.016	1.427	38.922	1.802	36.388	119.50	14.80	20:27
yeast3	37.898	1.125	38.868	1.015	43.639	1.154	43.666	113.00	13.17	25:22

Table 6.8: Classification results of QUEST algorithm.

Problem	10-fold CV error	test set error	no. leaf nodes
abalone1	72.89	73.88	20
abalone2	73.79	75.12	14
abalone3	73.82	72.63	12
australian1	13.54	11.56	2
australian2	15.47	17.34	2
australian3	15.47	11.56	2
balance1	23.50	17.83	16
balance2	18.38	19.75	13
balance3	19.87	19.11	19
cmc1	45.20	45.26	9
cmc2	47.10	42.55	18
cmc3	45.47	39.30	11
concentric1	3.47	2.88	25
concentric2	3.09	2.56	29
concentric3	2.67	2.88	29
diabetes1	25.87	23.44	2
diabetes2	25.35	24.48	2
diabetes3	25.87	24.48	2
german1	26.93	28.40	4
german2	27.07	28.80	4
german3	30.00	30.00	1
monks1	8.06	11.11	9
monks2	33.14	28.24	31
monks3	6.56	2.78	3
satimage1	14.01	15.53	75
satimage2	15.13	15.71	33
satimage3	14.78	15.09	64
segment1	6.35	5.19	39
segment2	5.60	5.54	31
segment3	5.60	7.79	26
smoking1	30.42	30.63	1
smoking2	30.37	30.77	1
smoking3	49.40	30.77	1
spirals1	35.05	24.84	34
spirals2	18.21	25.05	57
spirals3	30.93	34.03	35
titanic1	22.06	20.51	3
titanic2	21.52	19.78	8
titanic3	21.58	20.15	5
vehicle1	31.55	26.89	28
vehicle2	29.65	31.13	61
vehicle3	31.23	28.77	20
yeast1	44.47	40.97	24
yeast2	43.85	42.32	14
yeast3	45.28	43.67	21

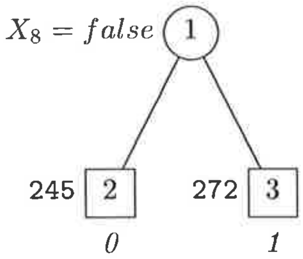


Figure 6.11: Decision tree obtained by QUEST for the **australian** data set.

The decision trees for the **monks** problems are shown in Figures 6.14 and 6.15. For the **monks1** problem, QUEST had no problem learning jacket_colour = red, but was confused by head_shape = body_shape. At node 8 it chose X4 as the split variable, which is unrelated to the class variable. Apart from this and some other bad decisions, it almost learned the concept fully. QUEST performed poorly on **monks2**, which being a parity-style problem is really quite difficult. For the **monks3** problem, QUEST learned the main part of the concept, (jacket_colour(X5) != blue AND body_shape(X2) != octagon), but did not pick up on the under-represented (jacket_colour(X5) = green AND holding(X4) = sword).

The trees generated for the **segment** data sets are quite complicated, and are generally beyond the immediate comprehension of the user. The tree for **segment1** is shown in Figure 6.16.

The trees obtained by QUEST for the first two permutations of the **titanic** data are shown in Figure 6.17. The simpler tree for **titanic1** shown in Figure 6.17(a) indicates that the average passenger did not survive if he/she was male or in third class. This is not quite intuitive, because it means that male children from first class died. For **titanic2**, the tree shown in Figure 6.17(b) agrees with the previous, but now admits that first and second class boys survived, as well as girls from third class.

Another useful output of the QUEST program is a ranking of the variables according to their importance for discrimination. Note that this ranking is not necessarily reflected in the final tree. That is, the variables that are most important according to the ranking may not be included in the tree due to masking by other variables. The ranking of variables for some of the data sets are listed below. In the tables, the most important variable has rank 100 and all others are normalised with respect to it. A description of the ranking method is found in Section 2.8.1.

The ranking of variables for **cmc1** is shown in Table 6.9. Age, number of children and education are ranked as the top three variables for this problem, while religion is ranked second-to-last. The ranking for **segment1** is shown in Table 6.10.

Table 6.9: Variable importance measure generated by QUEST for **cmc1**.

Variable	X1	X2	X3	X4	X5	X6	X7	X8	X9
Importance	100	77	25	99	37	8	51	53	40

Table 6.10: Variable importance measure generated by QUEST for **segment1**.

Variable	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
Importance	24	84	4	6	12	8	30	20	100	52	83

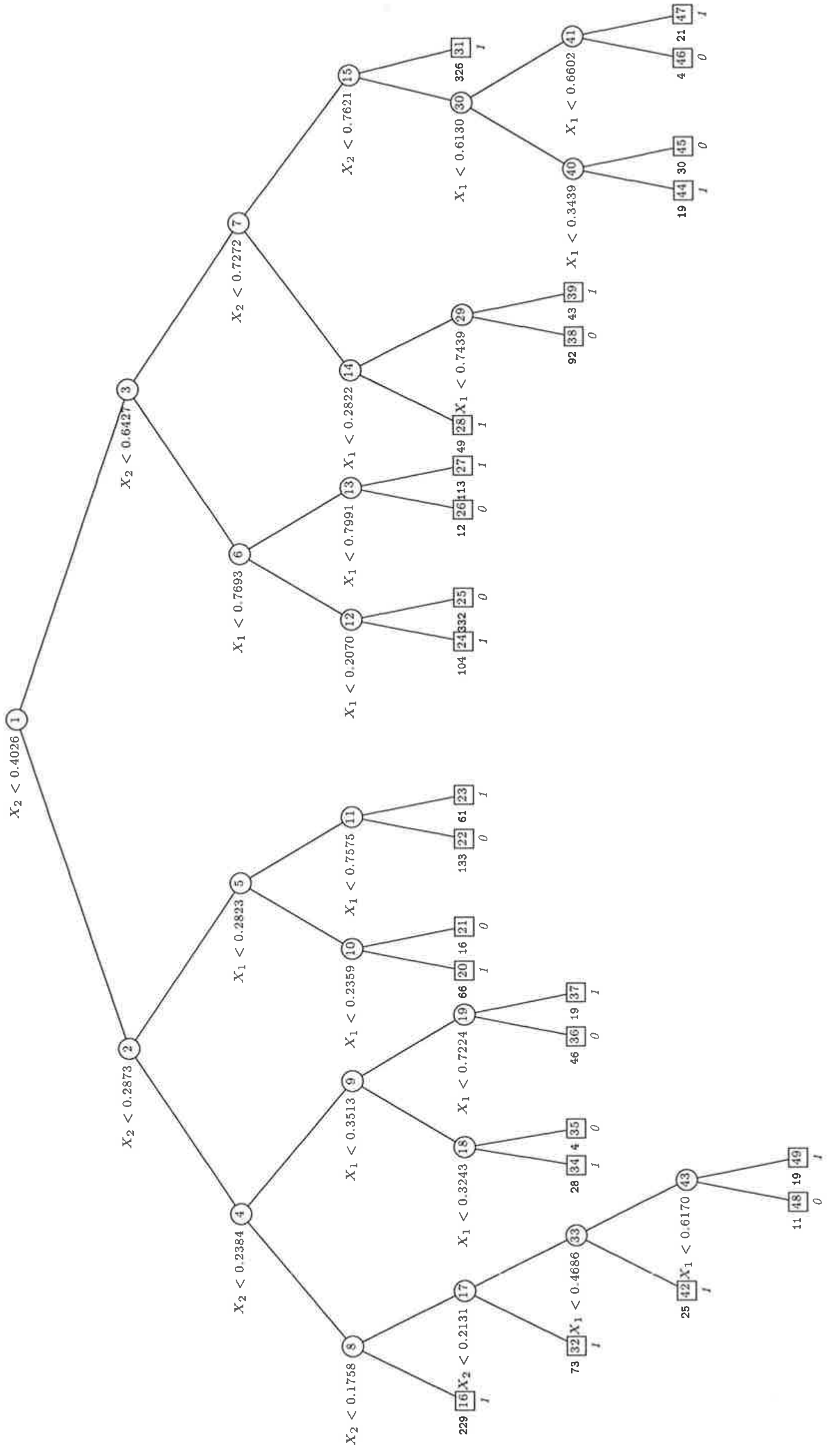


Figure 6.12: Decision tree obtained by QUEST for **concentric1**.

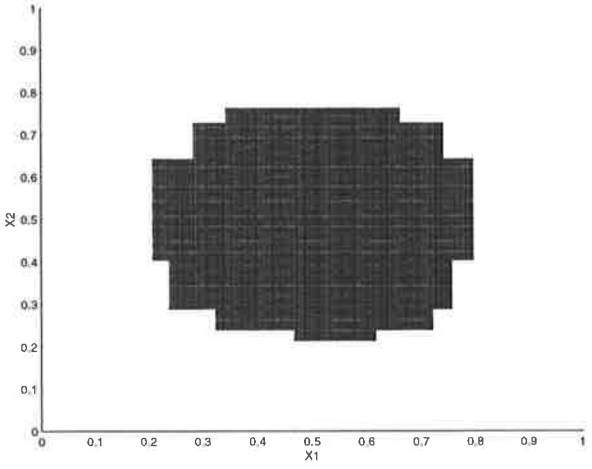


Figure 6.13: The decision regions created by QUEST for **concentric1**.

The variable importance measures for the **monks** problems are shown in Table 6.11. For **monks1**, QUEST has identified the importance of X1, X2 and X5, but has erroneously given X4 a high rank as well. For **monks2**, all attributes have the same conceptual importance; yet X5 has been given almost double the rank of the next highest variable. For **monks3**, X2 and X5 have been clearly identified as the most important variables, but X4 missed out due to the under-representation of this part of the concept in the training data.

Table 6.11: Variable importance measure generated by QUEST for **monks** problems.

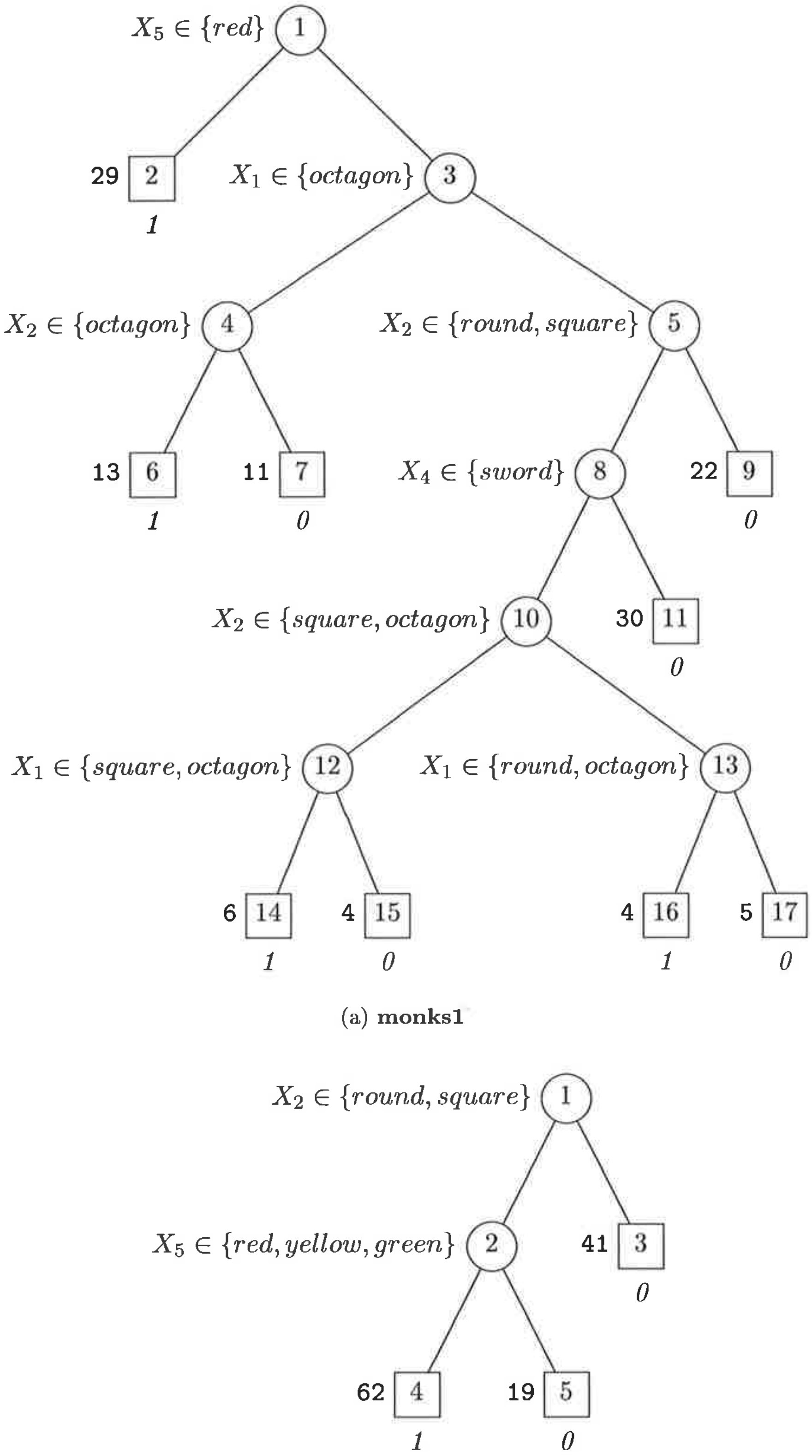
Variable	X1	X2	X3	X4	X5	X6
monks1	100	67	19	73	72	14
monks2	55	46	65	39	100	41
monks3	3	72	3	1	100	6

6.7 Experimental Results of Simple Methods

The results of the simple classifiers are shown in Table 6.12. The kNN and GLIM algorithms involved parameter selection, so the minimum validation set error, optimal value of k and best number of epochs are shown. For the other methods there were no parameters to select, so only the test set error is shown. The execution time is given in hours:minutes for the kNN and GLIM methods. The other algorithms took on the order of seconds and were only run once, so the time is not recorded.

On average, the kNN algorithm achieved the lowest classification error on the test set, and was only surpassed significantly by the ML classifier on **australian**, **balance** and **vehicle**. The disadvantage of the kNN classifier is that classification of a new sample is $\mathcal{O}(n_{tr}.d)$, and the training samples must be stored to perform classification. The ML classifier was the next best method on average over the problems. Strangely, the ML classifier performed poorly on **segment1** but well on **segment2** and **segment3**.

Since the GLIM classifier did not achieve 0% error on any of the data sets, none of them are linearly separable. For **vehicle**, however, the GLIM had lower errors than kNN, and similar errors for **german**. The performance of the GLIM classifier was relatively sensitive


$X_5 \in \{red\}$ 1
29 2
1
$X_1 \in \{octagon\}$ 3
$X_2 \in \{octagon\}$ 4
$X_2 \in \{round, square\}$ 5
13 6
1
11 7
0
$X_4 \in \{sword\}$ 8
22 9
0
$X_2 \in \{square, octagon\}$ 10
30 11
0
$X_1 \in \{square, octagon\}$ 12
$X_1 \in \{round, octagon\}$ 13
6 14
1
4 15
0
4 16
1
5 17
0
$X_2 \in \{round, square\}$ 1
$X_5 \in \{red, yellow, green\}$ 2
41 3
0
62 4
1
19 5
0

(a) **monks1**

(b) **monks3**

Figure 6.14: Decision Trees obtained for the **monks** problems using QUEST.

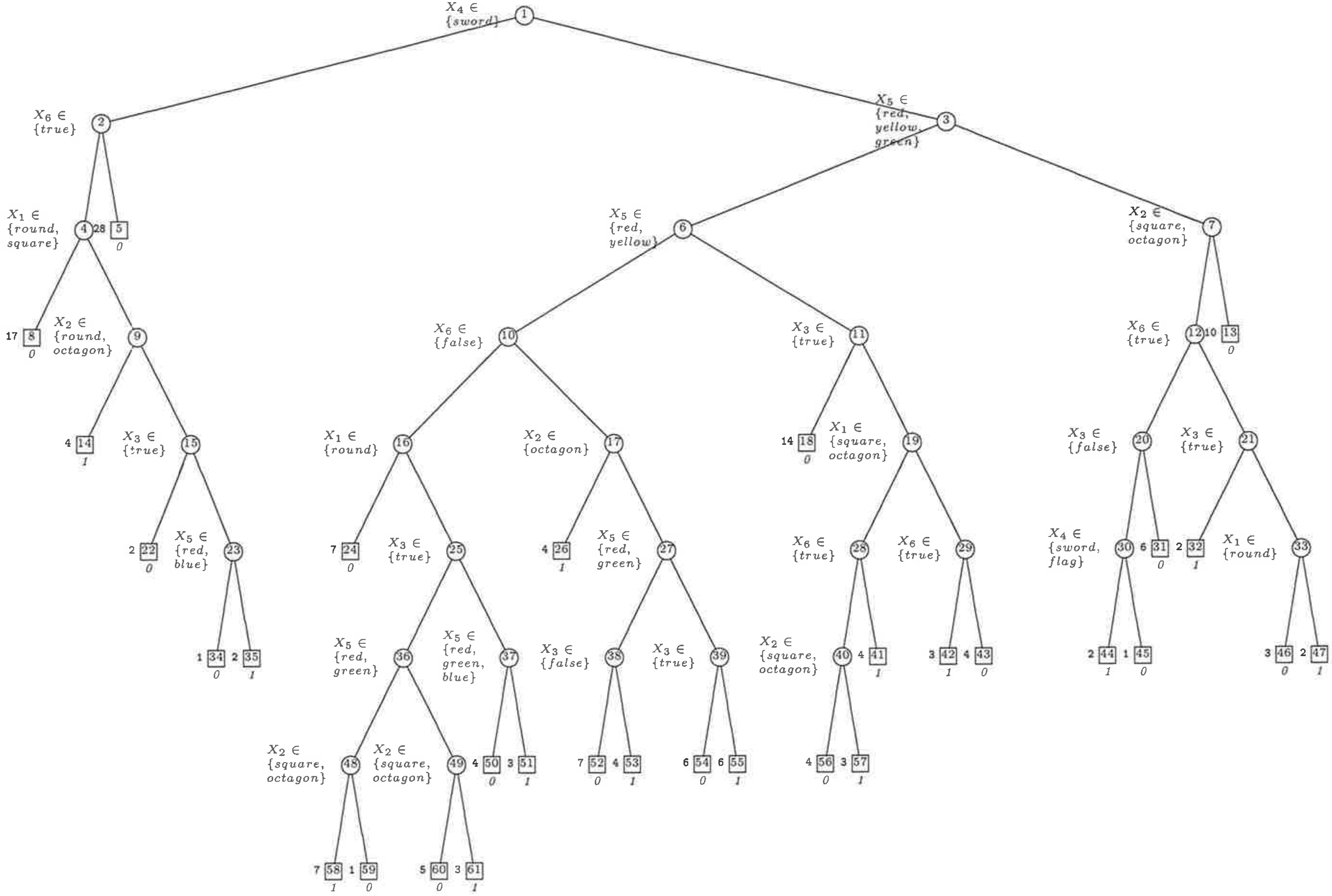


Figure 6.15: Decision Tree obtained for **monks2** using QUEST.

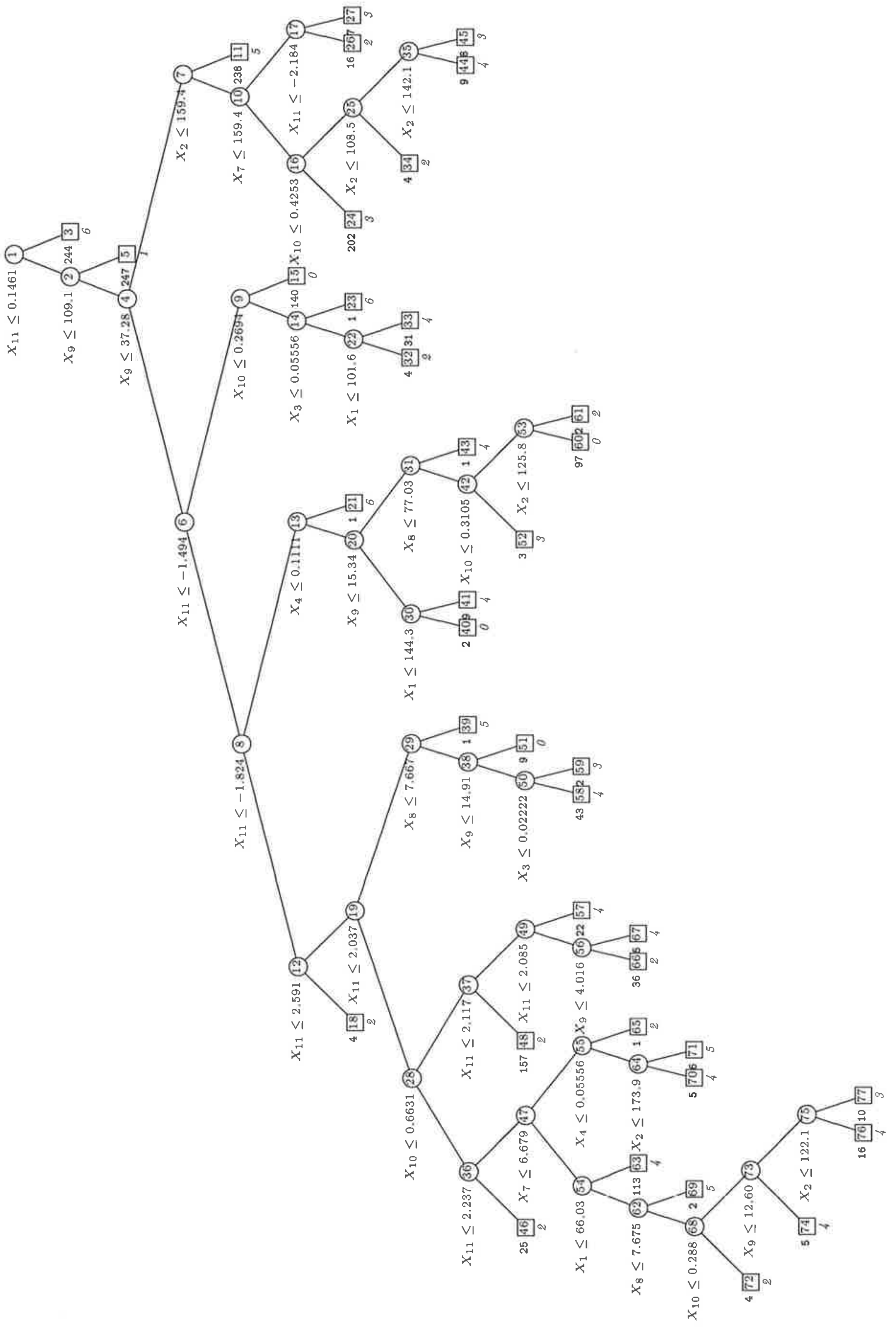
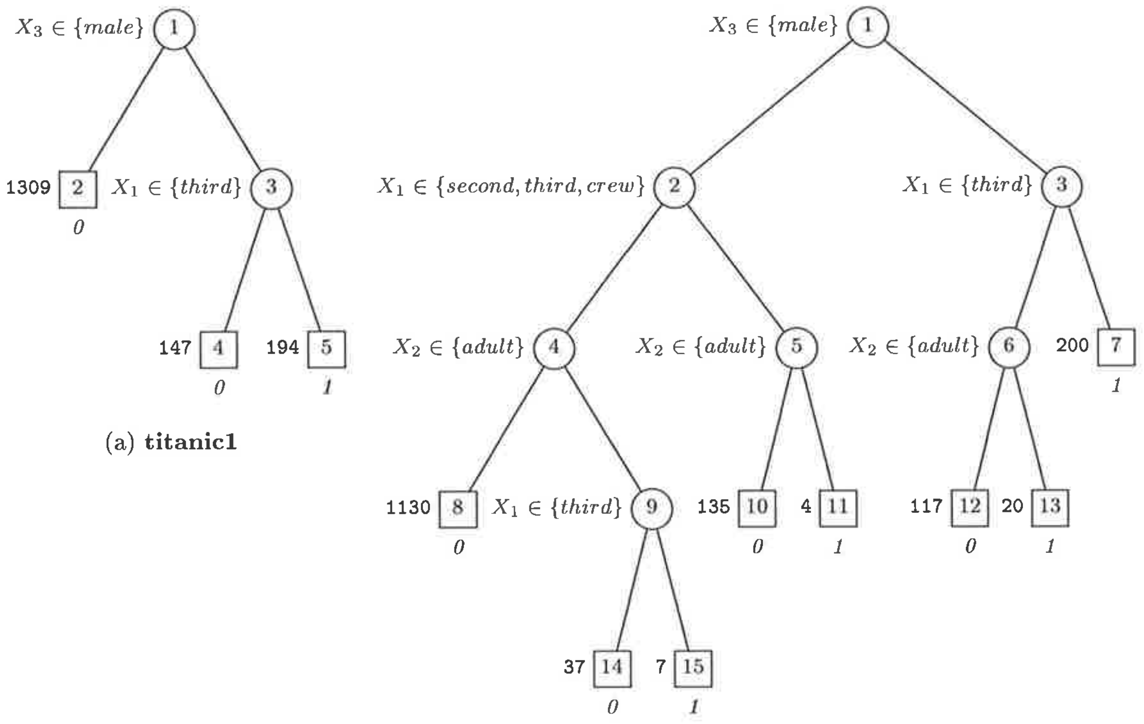


Figure 6.16: Decision Tree obtained for **segment1** problem using QUEST.

Table 6.12: Classification error rates for the simple classifiers: k-Nearest Neighbours, Generalised Linear Machine, Minimum-Distance-to-Means, Parallelepiped and Gaussian Maximum Likelihood.

	kNN				GLIM				MDTM	PPD	ML
problem	k	val.%	test%	t(h:m)	ep.	val.%	test%	t(h:m)	test%	test%	test%
abalone1	47	72.13	74.45	49:38	19	78.16	84.79	14:37	91.39	90.24	99.90
abalone2	125	72.70	76.08	50:26	136	75.00	77.51	19:26	92.06	92.92	99.90
abalone3	25	72.13	75.50	50:25	18	74.81	88.33	15:18	91.48	93.68	99.90
australian1	16	26.74	32.95	0:03	56	25.00	34.68	0:13	37.57	39.88	21.97
australian2	4	30.23	33.53	0:03	364	26.74	36.99	0:13	38.73	39.31	20.81
australian3	6	30.81	31.21	0:03	829	27.33	27.75	0:13	31.79	36.99	16.18
balance1	42	10.26	10.83	0:01	661	12.82	8.28	0:6	22.29	54.14	5.73
balance2	21	9.62	8.92	0:01	708	14.74	10.19	0:6	22.93	54.14	5.73
balance3	36	8.97	10.83	0:01	83	8.97	21.02	0:6	24.84	54.14	7.64
cmc1	48	42.12	46.34	0:50	128	52.72	63.42	0:32	59.35	64.23	44.99
cmc2	15	45.38	44.99	0:50	7	52.17	73.98	0:32	63.96	65.04	52.30
cmc3	62	46.20	40.65	0:50	4	51.63	72.36	0:33	59.89	66.67	42.01
concentric1	65	0.64	1.44	16:27	1	36.80	36.96	0:21	49.92	23.52	4.32
concentric2	11	0.96	0.64	16:19	1	36.96	36.80	0:21	51.04	22.88	3.68
concentric3	58	0.96	1.12	12:14	5	36.80	36.96	0:21	51.52	23.04	4.48
diabetes1	7	25.00	25.00	0:05	645	32.29	64.58	0:10	38.02	32.29	24.48
diabetes2	7	23.96	27.60	0:05	544	29.17	32.29	0:10	33.85	34.38	27.60
diabetes3	17	25.52	20.31	0:05	292	33.85	32.29	0:10	37.50	64.58	27.08
german1	29	28.80	30.00	0:19	1	30.00	30.00	0:28	40.80	67.20	36.40
german2	19	26.80	31.20	0:15	1	70.00	30.00	0:28	41.20	32.00	28.80
german3	21	29.20	28.40	0:16	114	29.60	30.00	0:28	37.20	28.40	29.20
monks1	1	14.29	15.51	0:01	9	30.95	44.91	0:02	34.03	33.33	27.08
monks2	4	28.57	25.00	0:01	4	33.93	32.87	0:03	46.53	32.87	25.69
monks3	11	9.76	8.80	0:01	47	17.07	21.53	0:02	19.44	52.78	11.34
satimage1	4	9.63	9.20	145:40	460	22.56	23.31	10:07	23.91	34.53	14.66
satimage2	4	8.89	9.07	75:23	67	21.32	31.57	10:09	23.66	37.95	14.47
satimage3	4	10.69	6.84	75:55	89	22.93	23.68	9:49	20.31	33.23	12.55
segment1	1	11.96	8.82	11:58	906	24.78	38.58	2:16	37.37	17.47	85.64
segment2	1	10.75	8.82	18:46	837	26.69	34.78	2:17	33.22	15.74	14.88
segment3	1	10.23	10.21	11:31	677	22.36	40.83	2:15	36.33	16.26	19.20
smoking1	51	30.25	30.95	27:16	133	33.33	34.73	1:19	64.06	88.53	65.59
smoking2	32	30.39	29.97	43:00	1	30.39	31.37	1:19	62.80	88.95	63.50
smoking3	49	29.83	31.09	27:38	4	44.82	38.80	1:20	65.87	89.09	65.45
spirals1	1	7.22	0.42	0:01	100	41.24	60.63	0:2	51.57	50.31	48.43
spirals2	1	7.22	2.08	0:01	2	42.27	43.96	0:2	51.57	50.52	51.77
spirals3	1	8.25	1.46	0:01	31	43.30	48.13	0:2	51.36	51.57	51.15
titanic1	3	20.55	20.15	9:09	1	21.82	21.60	0:17	30.67	32.49	21.23
titanic2	3	22.18	20.15	8:32	1	26.55	25.59	0:17	32.12	32.49	20.69
titanic3	6	22.55	21.60	8:45	1	67.64	67.70	0:17	32.30	32.30	22.87
vehicle1	1	33.18	37.26	15:55	229	32.23	29.25	0:30	60.38	62.26	12.74
vehicle2	6	32.23	38.68	15:21	345	29.86	34.91	0:30	60.38	67.45	18.40
vehicle3	3	31.75	36.79	14:31	544	30.81	23.59	0:30	63.21	65.57	14.15
yeast1	20	38.28	42.32	1:59	29	55.80	65.50	1:04	46.09	83.83	69.00
yeast2	41	39.89	37.47	1:59	3	49.33	80.32	1:04	48.52	79.78	68.73
yeast3	13	40.16	42.05	2:00	13	45.82	58.22	1:04	49.60	77.90	68.46



(a) **titanic1**

(b) **titanic2**

Figure 6.17: Decision Trees obtained for first two permutations of **titanic** using QUEST.

to the permutation of the data, while the other classifiers had similar errors for the three permutations. This is not surprising, because the GLIM is the only classifier in the table that relies on the order of the training samples.

The kNN classifier achieved almost zero error on the **spirals** data set with $k = 1$, which is due to the large distance between the two spiral arms. For the **concentric** problem, the ML classifier had the perfect density functions, but still made some errors because the density parameters were estimated from finite data. The kNN classifier did not manage so well with discrete data sets like **balance** and **monks**, which nevertheless can be perfectly classified.

6.8 Results of Comparison

This section compares the best-case performance of all classification methods examined, the average-case performance of EPrep and the MLP, the understandability of EPrep's features and QUEST's decision trees, and the computational complexity of the algorithms used.

6.8.1 Best-case Performance Comparison

Table 6.13 shows the test set error for each method on each data set. For each problem, the lowest test set error is shown in bold-face. Table 6.14 shows how many times each of the EPrep, MLP, QUEST, kNN and ML classifiers achieved the lowest, second-lowest or third-lowest test set error. The following observations can be made from Tables 6.13 and 6.14:

- The simpler methods to the right of the double vertical line performed relatively poorly on average, except for kNN.

Table 6.13: Classification results of all methods.

Problem	EPrep	MLP	QUEST	kNN	GLIM	MDTM	PPD	ML
abalone1	75.60	**73.30**	73.88	74.45	84.79	91.39	90.24	99.90
abalone2	**73.97**	75.41	75.12	76.08	77.51	92.06	92.92	99.90
abalone3	74.64	75.31	**72.34**	75.50	88.33	91.48	93.68	99.90
australian1	**10.40**	13.87	11.56	32.95	34.68	37.57	39.88	21.97
australian2	**16.76**	17.34	17.34	33.53	36.99	38.73	39.31	20.81
australian3	**10.98**	**10.98**	11.56	31.21	27.75	31.79	36.99	16.18
balance1	**0.00**	7.64	17.83	10.83	8.28	22.29	54.14	5.73
balance2	**0.00**	1.27	19.75	8.92	10.19	22.93	54.14	5.73
balance3	**0.64**	2.55	19.11	10.83	21.02	24.84	54.14	7.64
cmc1	40.92	**40.65**	45.26	46.34	63.42	59.35	64.23	44.99
cmc2	44.99	45.26	**42.55**	44.99	73.98	63.96	65.04	52.30
cmc3	**38.21**	39.02	39.30	40.65	72.36	59.89	66.67	42.01
concentric1	**0.00**	1.12	2.88	1.44	36.96	49.92	23.52	4.32
concentric2	1.92	1.44	2.56	**0.64**	36.80	51.04	22.88	3.68
concentric3	1.76	1.28	2.88	**1.12**	36.96	51.52	23.04	4.48
diabetes1	**22.92**	**22.92**	23.44	25.00	64.58	38.02	32.29	24.48
diabetes2	24.48	**22.40**	24.48	27.60	32.29	33.85	34.38	27.60
diabetes3	24.48	20.83	24.48	**20.31**	32.29	37.50	64.58	27.08
german1	26.80	**25.60**	28.40	30.00	30.00	40.80	67.20	36.40
german2	27.20	**24.00**	28.80	31.20	30.00	41.20	32.00	28.80
german3	28.40	**22.40**	30.00	28.40	30.00	37.20	28.40	29.20
monks1	**0.00**	**0.00**	11.11	15.51	44.91	34.03	33.33	27.08
monks2	0.93	**0.00**	28.24	25.00	32.87	46.53	32.87	25.69
monks3	2.78	**0.46**	2.78	8.80	21.53	19.44	52.78	11.34
satimage1	21.68	13.80	15.53	**9.20**	23.31	23.91	34.53	14.66
satimage2	16.46	12.99	15.71	**9.07**	31.57	23.66	37.95	14.47
satimage3	15.34	11.19	15.09	**6.84**	23.68	20.31	33.23	12.55
segment1	**4.84**	5.19	5.19	8.82	38.58	37.37	17.47	85.64
segment2	4.50	**3.46**	5.54	8.82	34.78	33.22	15.74	14.88
segment3	**6.40**	7.61	7.79	10.21	40.83	36.33	16.26	19.20
smoking1	31.19	30.67	**30.63**	30.95	34.73	64.06	88.53	65.59
smoking2	30.91	30.81	30.77	**29.97**	31.37	62.80	88.95	63.50
smoking3	31.47	31.09	**30.77**	31.09	38.80	65.87	89.09	65.45
spirals1	24.22	45.83	24.84	**0.42**	60.63	51.57	50.31	48.43
spirals2	34.66	50.42	25.05	**2.08**	43.96	51.57	50.52	51.77
spirals3	32.15	46.25	34.03	**1.46**	48.13	51.36	51.57	51.15
titanic1	**20.15**	20.69	20.51	**20.15**	21.60	30.67	32.49	21.23
titanic2	20.15	20.15	**19.78**	20.15	25.59	32.12	32.49	20.69
titanic3	20.69	21.60	**20.15**	21.60	67.70	32.30	32.30	22.87
vehicle1	13.68	19.34	26.89	37.26	29.25	60.38	62.26	**12.74**
vehicle2	**17.45**	22.64	31.13	38.68	34.91	60.38	67.45	18.40
vehicle3	16.51	18.87	28.77	36.79	23.59	63.21	65.57	**14.15**
yeast1	46.90	41.24	**40.97**	42.32	65.50	46.09	83.83	69.00
yeast2	47.71	**36.39**	42.32	37.47	80.32	48.52	79.78	68.73
yeast3	52.56	43.67	43.67	**42.05**	58.22	49.60	77.90	68.46

Table 6.14: Frequency of top ranking for the best five classification methods.

Classifier	EPrep	MLP	QUEST	kNN	ML
number of times first	15	13	7	12	2
number of times first or second	31	31	21	17	4
number of times first, second or third	36	42	30	27	17

- The methods that most often achieved the lowest test set error were EPrep (15 times), the MLP (13 times), kNN (12 times) and QUEST (7 times).
- The MLP was among the top three classifiers for 42 data sets, EPrep for 36, and QUEST for 30. In this sense, the MLP was the most reliable classifier.
- For the synthetic problems **balance**, **concentric**, **monks** and **spirals** there is no class overlap and zero error is attainable. The relatively simple **balance** problem proved too difficult for the MLP and QUEST, but was solved by EPrep. Finite but small errors were obtained for all methods on **concentric** due to the limited amount of data. Both EPrep and the MLP were able to perfectly learn the concept in **monks1**, while only the MLP was able to learn **monks2**. None of the methods could cope with the noise and under-representation of the concept in **monks3**. The **spirals** problem was especially difficult for the MLP, and still difficult for EPrep and QUEST, while kNN achieved almost zero error on this data set. Note that in other studies of the **spirals** problem (Lang and Whitbrock, 1989; Koza, 1992*a*) zero error has been achieved. In the experiments presented here, however, only a limited portion of the data was used for training which made learning more difficult.
- The classifier achieving best performance was not always the same for the three permutations. Nevertheless, the ascendancy of one algorithm over the others for at least two out of the three permutations was observed for all problems except **abalone**, **cmc** and **yeast**.
- **smoking** was the only data set for which the lowest test set error was not significantly less than the default error rate.

The variability of the best classifier with permutation of the data demonstrates the value of using more than one permutation of the data. For example, if examination **diabetes** had been based only on the third permutation, it would have appeared that the kNN was most suited to this data set.

It is not sufficient to state that method A achieved superior classification performance to method B on the test set because its test set error was lower. For example, the test set for the **balance** problem contains 157 samples, so the difference of 1.27% between the errors of EPrep and the MLP equates to 2 samples. A statistical comparison must be performed to ensure that this difference did not come about by chance. The testing method used here is the one-way repeated measures design described in Section 2.11.3. The simple methods to the right of the double vertical line generally performed poorly, so the statistical comparison was constrained to the first three methods. Since $k = 3$, $k^* = 3(3-1)/2 = 3$. The Z statistic values listed in (Marascuilo and McSweeney, 1977) only go up to 120 degrees of freedom; since all data sets examined had more than 121 test samples, $\nu = \infty$ was used. From the table, $Z^{\infty}_{3:0.975} = 2.39$ is the value used to obtain 95% confidence intervals, because the test is two-tailed. The 95% confidence intervals of differences in test set classification error rate (%) for each data set are given in Table 6.15. Intervals that include 0 do not indicate a significant difference. Those differences that are significant at the $\alpha = 0.05$ level are high-lighted in bold.

The following observations can be made from Table 6.15:

- Of the 45 comparative experiments, only 18 yielded a statistically significant result.
- EPrep achieved a lower test set error than the MLP and QUEST on **balance1** only.
- The MLP achieved a lower test set error than EPrep and QUEST on 4 data sets, **monks3** and all permutations of **satimage**.
- QUEST achieved a lower test set error than EPrep and the MLP for **spirals2** only.
- The MLP beat EPrep on 6 data sets, while EPrep beat the MLP on 4.
- EPrep beat QUEST on 9 data sets, while QUEST beat EPrep on 3.
- The MLP beat QUEST on 13 data sets, while QUEST beat the MLP on 3.
- EPrep performed relatively poorly on the **satimage** and **yeast** problems. This is most likely due to the relatively high dimensionality of the **satimage** data.

Based on these results, we can loosely say that, in terms of classification accuracy, the MLP was one-and-a-half times as accurate as EPrep, and EPrep was three times as accurate as QUEST.

6.8.2 Average-case Performance Comparison

The mean and standard deviation of test set errors achieved by EPrep and the MLP over the 10 runs are compared in Table 6.16. The sixth column contains the difference between the average percentage error obtained by EPrep and that obtained by the MLP. A Student's t-test for the difference between means of samples from distributions with unequal variances was performed to test for the significance of the differences in mean error; the test is fully described in Section 2.11.2. The last column in Table 6.16 contains the probability $p = 1 - \alpha$ that the difference between the means did not arise by chance. Those differences that are significant at a level of $\alpha = 0.05$ are high-lighted in bold.

EPrep generally had a higher standard deviation of test set errors than the MLP. In some instances, such as **balance**, **diabetes**, **smoking** and **yeast**, the standard deviation of EPrep's errors was several times larger than that of the MLP. Note that this was the case even when the best run of EPrep was superior to the best run of the MLP, such as for **balance1** and **spirals**. The t-test revealed that for 27 of the 45 data sets the error difference was significant. Of those 27 cases, the MLP obtained a lower mean error 15 times. While this is not an overwhelming majority, the MLP was lower than EPrep by a larger margin on average. Therefore it can be concluded that the MLP was more robust from run to run than EPrep.

6.8.3 Comparison of Interpretability

There are two types of auxiliary information that can be provided by the QUEST and EPrep algorithms: identification of the measurements that are of most use for classification, and simple relationships between the variables that reveal structure in the data. While the other classification methods may yield better accuracy, they are not considered to offer any useful information for interpretation of the data. The practice of identifying those measurements

Table 6.15: Confidence intervals for the differences in percentage test set errors of EPrep, the MLP and Quest.

problem	MLP - EPrep	QUEST - EPrep	QUEST - MLP
abalone1	[-6.23, 1.64]	[-5.66, 2.21]	[-3.36, 4.51]
abalone2	[-2.48, 5.35]	[-2.77, 5.07]	[-4.20, 3.63]
abalone3	[-3.12, 4.46]	[-5.80, 1.78]	[-6.47, 1.11]
australian1	[-1.94, 8.88]	[-4.25, 6.57]	[-7.72, 3.10]
australian2	[-4.71, 5.87]	[-4.71, 5.87]	[-5.29, 5.29]
australian3	[-4.22, 4.22]	[-3.64, 4.80]	[-3.64, 4.80]
balance1	**[0.61, 14.67]**	**[10.80, 24.87]**	**[3.16, 17.22]**
balance2	[-5.65, 8.19]	**[12.82, 26.67]**	**[11.55, 25.39]**
balance3	[-5.23, 9.05]	**[11.33, 25.61]**	**[9.42, 23.70]**
cmc1	[-6.53, 5.99]	[-1.92, 10.59]	[-1.65, 10.86]
cmc2	[-5.69, 6.23]	[-8.40, 3.52]	[-9.01, 3.59]
cmc3	[-5.10, 6.73]	[-4.83, 7.00]	[-5.64, 6.18]
concentric1	[-0.31, 2.55]	**[1.45, 4.31]**	**[0.33, 3.19]**
concentric2	[-1.98, 1.02]	[-0.86, 2.14]	[-0.38, 2.62]
concentric3	[-2.10, 1.14]	[-0.50, 2.74]	[-0.02, 3.22]
diabetes1	[-6.59, 6.59]	[-6.07, 7.11]	[-6.07, 7.11]
diabetes2	[-7.83, 3.67]	[-5.75, 5.75]	[-3.67, 7.83]
diabetes3	[-9.30, 2.01]	[-5.66, 5.66]	[-2.01, 9.30]
german1	[-7.91, 5.51]	[-5.11, 8.31]	[-3.87, 9.47]
german2	[-10.09, 3.69]	[-5.29, 8.49]	[-2.09, 11.69]
german3	[-13.15, 1.15]	[-5.55, 8.75]	**[0.62, 14.58]**
monks1	[-3.13, 3.13]	**[7.98, 14.24]**	**[7.98, 14.24]**
monks2	[-5.98, 4.12]	**[22.26, 32.37]**	**[23.19, 33.29]**
monks3	**[-3.74, -0.89]**	[-1.43, 1.43]	**[0.89, 3.74]**
satimage1	**[-10.18, -5.59]**	**[-6.46, -1.86]**	**[1.43, 6.02]**
satimage2	**[-5.50, -1.46]**	[-2.77, 1.28]	**[0.71, 4.75]**
satimage3	**[-6.15, -2.18]**	[-2.23, 1.74]	**[1.93, 5.90]**
segment1	[-2.09, 2.78]	[-1.57, 3.30]	[-1.92, 2.95]
segment2	[-3.17, 1.10]	[-1.10, 3.17]	[-0.06, 4.21]
segment3	[-1.53, 3.95]	[-1.36, 4.13]	[-2.57, 2.92]
smoking1	[-1.33, 0.21]	[-1.33, 0.21]	[-0.77, 0.77]
smoking2	[-0.86, 0.86]	[-1.00, 0.72]	[-1.00, 0.72]
smoking3	[-1.48, 0.64]	[-1.76, 0.36]	[-1.34, 0.78]
spirals1	**[14.71, 28.71]**	[-6.37, 7.62]	**[-28.08,-14.09]**
spirals2	**[8.53, 23.20]**	**[-16.94, -2.27]**	**[-32.80,-18.14]**
spirals3	**[7.28, 20.69]**	[-4.83, 8.59]	**[-18.96, -5.26]**
titanic1	[-0.52, 1.61]	[-0.70, 1.43]	[-1.24, 0.88]
titanic2	[-2.45, 2.45]	[-2.82, 2.09]	[-2.82, 2.09]
titanic3	[-1.33, 3.15]	[-2.78, 1.70]	[-3.69, 0.79]
vehicle1	[-2.10, 13.42]	**[5.45, 20.96]**	[-0.21, 15.30]
vehicle2	[-2.78, 13.16]	**[5.71, 21.65]**	**[0.52, 16.46]**
vehicle3	[-5.67, 10.38]	**[4.24, 20.29]**	**[1.88, 17.93]**
yeast1	[-11.99, 0.67]	[-12.26, 0.40]	[-6.94, 6.40]
yeast2	**[-17.29, -5.35]**	[-11.36, 0.58]	[-1.15, 13.01]
yeast3	**[-15.07, -2.72]**	**[-15.07, -2.72]**	[-6.59, 6.59]

Table 6.16: Average percentage test set errors of EPrep and MLP.

Problem	EPrep		MLP		EPrep - MLP	$1-\alpha$
	μ	σ	μ	σ	(%)	
abalone1	76.81	1.50	75.809	1.605	1.00	0.833
abalone2	76.62	1.29	77.321	1.100	-0.70	0.792
abalone3	77.51	2.01	76.172	1.482	1.33	0.891
australian1	11.39	0.37	13.468	0.820	**-2.07**	1.000
australian2	17.17	0.37	18.208	0.734	**-1.03**	0.998
australian3	11.50	0.31	11.272	0.734	0.22	0.618
balance1	4.90	8.00	7.134	1.964	-2.23	0.590
balance2	8.60	9.99	7.643	2.383	0.95	0.226
balance3	14.33	15.75	7.389	2.788	6.94	0.800
cmc1	44.63	3.21	43.117	2.074	1.51	0.771
cmc2	45.85	1.83	46.748	1.926	-0.90	0.701
cmc3	42.87	3.22	41.463	2.649	1.40	0.700
concentric1	2.51	0.99	2.192	0.908	0.31	0.536
concentric2	2.22	0.73	2.192	0.735	0.02	0.067
concentric3	2.00	0.41	1.712	0.512	0.28	0.818
diabetes1	23.23	1.24	22.135	0.859	**1.09**	0.965
diabetes2	29.11	4.40	23.906	1.355	**5.20**	0.995
diabetes3	28.59	6.47	21.094	0.563	**7.49**	0.995
german1	29.68	2.73	26.920	1.969	**2.76**	0.981
german2	28.96	2.02	26.400	2.199	**2.56**	0.985
german3	27.48	1.53	20.960	1.375	**6.52**	1.000
monks1	7.36	7.63	0.371	0.897	**6.98**	0.983
monks2	7.96	8.77	22.82	15.80	**-14.86**	0.979
monks3	2.78	0.00	0.97	0.75	**1.81**	1.000
satimage1	26.17	6.15	14.413	0.427	**11.75**	0.999
satimage2	19.99	2.96	13.250	0.586	**6.74**	1.000
satimage3	17.06	1.82	12.293	0.791	**4.76**	1.000
segment1	6.83	1.21	6.26	1.66	0.57	0.607
segment2	6.21	1.48	5.29	1.88	0.92	0.760
segment3	7.23	0.75	8.49	0.82	**-1.26**	0.997
smoking1	52.27	27.00	30.67	0.09	**21.60**	0.970
smoking2	41.82	21.86	30.93	0.15	10.89	0.853
smoking3	49.66	28.50	30.88	0.15	18.78	0.936
spirals1	28.98	2.76	48.97	2.06	**-19.99**	1.000
spirals2	30.69	3.75	49.12	2.40	**-18.43**	1.000
spirals3	29.52	4.83	48.27	2.24	**-18.75**	1.000
titanic1	20.15	0.01	20.690	0.000	**-0.54**	1.000
titanic2	19.48	0.49	20.036	0.454	**-0.55**	0.983
titanic3	21.13	0.68	21.597	0.000	-0.46	0.942
vehicle1	13.77	0.86	21.651	1.548	**-7.88**	1.000
vehicle2	18.73	2.13	21.745	1.482	**-3.01**	0.998
vehicle3	15.71	1.61	17.830	2.666	**-2.12**	0.952
yeast1	57.82	7.93	40.296	1.166	**17.52**	1.000
yeast2	59.54	10.93	38.922	1.802	**20.62**	0.999
yeast3	53.02	1.36	43.639	1.154	**9.38**	1.000

in the final solution as exclusively useful is unwise, and is warned against in (Breiman *et al.*, 1984). The reason is that the measurements are generally not independent, so that some group of variables may in combination contain the same amount of information as some other group of variables.

The solution obtained by QUEST for **australian** (Figure 6.11) used the variable X8 only. While the solutions generated by EPrep were more complicated, the feature sets for the first and third permutations (Figures 6.4(a) and 6.4(b)) are quite small, and use X8 as well as X12 and X13. Hence there is some agreement between the two methods that X8 is an important variable. The decision tree for **segment1** (Figure 6.16) is much larger than the feature set generated by EPrep (Figure 6.7(c)). Among the variables highest in the tree and having the highest rank in Table 6.10 are X9, X2, X11 and X10. Of the 11 variables, EPrep's solution uses X2, X4, X5, X9, X10 and X11, with X9, X10 and X11 appearing more than once in the feature set. Both methods agreed upon the importance of X9, X2, X11 and X10, but not on X4 and X5 that have a low rank according to QUEST.

For the **monks1** problem, both QUEST (Figure 6.14(a)) and EPrep (Figure 6.3(a)) included a variable in the solution that was unassociated with the concept to be learned. In the case of EPrep, the presence of this variable did not interfere with the classification of the data, while QUEST's performance was degraded. For **monks2**, EPrep evolved a more compact solution (Figure 6.3(b)) than the QUEST tree (Figure 6.15), and the two solutions are quite different. The features evolved by EPrep for the **monks3** problem (Figure 6.3(c)) contained all the variables except for the useful X4; this excess confusion is probably the result of the noise in the training data. QUEST (Figure 6.14(b)) also missed the importance of X4, but used only the other critical variables.

Some consensus was also reached on the importance of X1, X2 and X4 for **cmc1**. The variable importance ranking produced by QUEST for **cmc1** (Table 6.9) has these three variables first, and gives a low rank to the subject's religion. The features evolved by EPrep for **cmc1** (Figure 6.7(a)) predominantly used X1, X2 and X4, and did not use the variable for religion.

The results for **concentric** (Figures 6.2(b) and 6.12) and **titanic** (Figures 6.6 and 6.17) reveal useful relationships in the data. While the tree of QUEST for **concentric** is large and complicated, and generally difficult to read, the solution presented by EPrep is succinct and understandable. The decision tree can only ever approximate the true decision boundary, whereas EPrep can produce an exact expression due to its versatility in combining different functions. Both methods gave similar results for **titanic**. Although the previously-known women-and-children-first rule was in operation, the analysis revealed that third class children were discriminated against. Both methods found that, for **titanic2**, third-class girls survived while third-class boys did not. This finding may have come about due to permutation of the data, and is not necessarily indicative of the whole data set. The features evolved by EPrep are slightly easier to read than the decision trees for this problem, because AND/OR statements are easier to translate into sentences.

6.8.4 Other Comparisons

There are other criteria by which to compare classification algorithms. The two addressed here are parameter selection and computational complexity. The more parameters there are associated with an algorithm, the less certain it is that the results obtained are the best that can be attained using the algorithm. On the other hand, a higher level of configurability means more flexibility, which can be an advantage. A lower computational complexity is always desirable, but more resources may be affordable if the algorithm can achieve significantly better results.

The simpler algorithms used in these experiments are the most appealing in terms of the number of parameters that need to be selected by the user. PPD, MDTM and ML have no parameters at all, and are deterministic. kNN is also deterministic, but requires the choice of k. The results of the GLIM algorithm depend on the choice of learning rate, maximum number of epochs, initial weights and order of the data samples. All of these simpler algorithms are quite easy to understand and to implement in a programming language. The EPrep, MLP and QUEST methods have more parameters to select, are harder to understand and are more prone to programming errors. EPrep is the most complicated algorithm of all, and has many different parameters to select. EPrep is also a stochastic algorithm, and the results depend on random choices during the algorithm apart from data sub-set selection, which is also randomised in QUEST. The MLP algorithm used trains in batch mode, so the order of training samples does not matter. The sample order was randomised internally by EPrep and QUEST.

A comparison of elapsed computation time is complicated by the multiplicity of platforms, and the variations in efficiency of coding of the different methods. Also, the concurrent load on Unix workstations can deceptively double or triple the time taken. The times quoted in this chapter are included only to give an idea of the relative times it would practically take to run each algorithm. A more rigorous analysis requires the computational complexity of the algorithm to be calculated.

The computation time of EPrep and the MLP were similar, being on the order of 1-10 hours, with EPrep generally taking longer. The QUEST algorithm took on the order of 1 minute, about two orders of magnitude faster than the other two methods. The simpler methods took on the order of less than one second, but for the kNN and GLIM methods an exhaustive search for the hyper-parameters was performed which took a long time.

The computational complexity of each algorithm for training and for classification of a single sample are shown in Table 6.17. Some of the complexities require further explanation. The kNN method requires no training. For the GLIM, e_{av} is the average number of epochs required for training. For QUEST, v is the fold of cross-validation, and d_{tree} is the depth of the tree obtained, which is proportional to the number of attributes and the complexity of the problem. For the MLP, e_{av} is the average number of epochs for training, and $n_{weights}$ is the number of weights in the network. For a two-hidden-layer network with n_1 nodes in the first hidden layer and n_2 in the second:

$$n_{weights} = (d+1).n_1 + (n_1+1).n_2 + (n_2+1).C$$

Therefore the number of weights involves a product of the number of hidden nodes with the number of classes and input variables. For EPrep, e_{opt} is the average number of fitness evaluations in the local optimisation routine, $\mathcal{O}_{gpp}$ is the complexity of the derived pre-processor, which will usually be proportional to d, and $\mathcal{O}_{cl}$ is the complexity of the classifiers used on the pre-processed data.

Of QUEST, the MLP and EPrep, QUEST is the most efficient, with EPrep the most expensive algorithm. The local optimisation used in EPrep is a severe burden on the computational load. Note that in the complexity of EPrep shown in Table 6.17, the cost of evaluating the pre-processing function is usually assumed to be much less than the cost of classification. If optimisation is not used, the complexity becomes $\mathcal{O}(G.M.\mathcal{O}_{cl})$, which can be less than the complexity for a neural network if $G.M.d < e_{av}.n_{weights}$. If, however, the average tree size or number of features in the population becomes large, the feature evaluation component of the fitness function can dominate computation time to a degree exponential with depth.

Although EPrep is more computationally intensive than the other methods, the evolved pre-processors and simple classifiers used require relatively few resources. Once the pre-processor has been evolved, it can be used for the problem independently of the EPrep algorithm. Therefore the algorithm may be preferable to the others according to the criterion

Table 6.17: Computational complexity of each classification algorithm for training, and for classification of a single sample.

algorithm	$\mathcal{O}(.)$ for training	$\mathcal{O}(.)$ for classifying
kNN	-	$n_{tr}.d$
GLIM	$n_{tr}.d.C.e_{av}$	$d.C$
MDTM	$n_{tr}.d$	$d.C$
PPD	$n_{tr}.d$	$d.C$
ML	$n_{tr}.d^2$	$d^2.C$
QUEST	$n_{tr}.d.v$	d_{tree}
MLP	$n_{tr}.e_{av}.n_{weights}$	$n_{weights}$
EPrep	$G.M.e_{opt}.(\mathcal{O}_{gpp} + \mathcal{O}_{cl})$	$\mathcal{O}_{gpp} + \mathcal{O}_{cl}$

that the final classifier have minimum complexity. For example, compare the results of EPrep and the MLP for the **abalone2** data set. The two algorithms achieved comparable test set error rates. The MLP has 1309 adaptive weights, which means that 1309 multiplications are required to classify a new sample. The classification system generated by EPrep has a pre-processor with 42 nodes which passes its features to the MDTM classifier. The pre-processor, shown in Figure 6.18, requires only 5 multiplications[1], and the classifier requires $d \times C = 232$ multiplications. Therefore EPrep's solution requires only about 18% of the computational and memory resources that the MLP does.

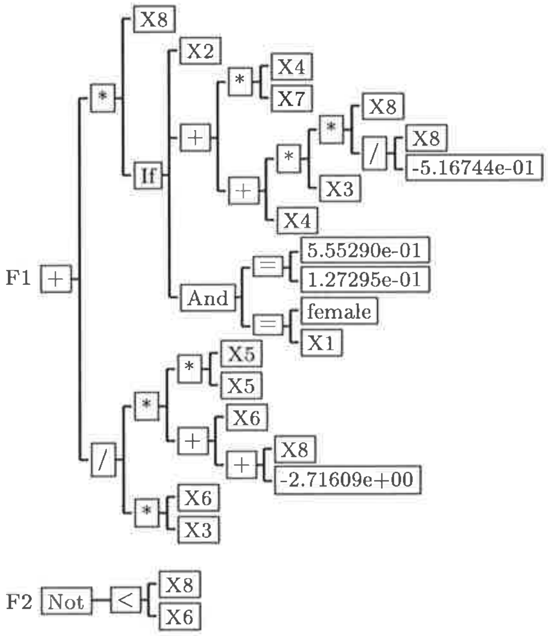


Figure 6.18: Best feature set evolved by EPrep for **abalone2**.

[1] Multiplications are the operations by which computational complexity is usually measured, because they take more time than additions. Divisions are counted as multiplications.

6.9 Postmortem of EPrep

In light of the comparative results, the problems for which EPrep performed relatively poorly or relatively well are identifiable. This section investigates the reasons why EPrep performed poorly or well on certain problems.

EPrep performed relatively well on the **abalone** problem, indicating that the algorithm does not have difficulty with a large number of classes. EPrep also excelled for the **australian** data set, which shows that the method is not sensitive to enumerated attributes with a large number of distinct values. EPrep's symbolic approach proved useful for **titanic** and for the synthetic problems, for which relatively good accuracy was achieved. In particular, EPrep classified the **balance** data relatively easily. The distinctive quality of this data set is that the attributes are discrete, but have a natural ordering. In absolute terms, however, the classification performance for **spirals** was quite bad. If the full data set were made available for training, a perfectly general solution may be attainable.

For **vehicle**, the ML classifier was already achieving the lowest error rate of all the classifiers other than EPrep. Consequently there was no way for EPrep to improve upon the classification rate other than to over-fit the data, therefore generalisation was poor. For the **smoking** problem, the best strategy was to always predict the most frequent class. Hence this problem is another candidate for over-fitting, since there is little else that can be done other than a very simple strategy. Nevertheless, EPrep managed to achieve almost-optimal error rates for this problem, even though it generated some enormously complicated pre-processors. This situation exemplifies the danger of interpreting results from machine learning algorithms: the user could look at the solution from EPrep and conclude that the data set is extremely complicated, whereas the decision tree generated by QUEST contains only one node.

The two data sets that were the most problematic for EPrep were **satimage** and **yeast**. The **satimage** data set contains the most attributes of all the data sets. EPrep's failure to effectively search the space of solutions for this data set is understandable, because of the size of the search space is a very-high-order polynomial in the number of input measurements for the problem (see Section 4.10). The reason for the poor performance on **yeast** is less obvious. Although this problem has 10 classes with disproportionate numbers of samples in the classes, the similar situation in **abalone** did not present a problem. From examination of the plots in Appendix E, the results were similar for **yeast1** and **yeast2**, but quite different for **yeast3**. In particular, the standard deviation of fitness, shown for convenience in Figure 6.19, displays quite different behaviour. The cause for this difference is that different parameters were used for the permutations. The common difference between the first two permutations and the third is that the PPD classifier was not used for **yeast3**. It is not obvious how the PPD classifier led to a higher initial diversity but lower final diversity in the population. The inability of EPrep to achieve competitive error rates on this data set suggests that, for the function and terminal sets chosen, representation of a useful pre-processor is not easy. The use of a different function set, or other modified parameters may be sufficient to improve the success of the algorithm.

6.10 Conclusion

This chapter has presented experiments designed to investigate the feasibility of a search for optimal generalised pre-processors for real-world problems, and to examine the advantages of this approach over existing methods for classification and feature extraction. The experiments have shown that EPrep is able to synthesise generalised pre-processors to improve the performance of a classifier for synthetic and real-world problems. Although the simple classifiers used by EPrep performed relatively poorly on their own, EPrep was able to significantly

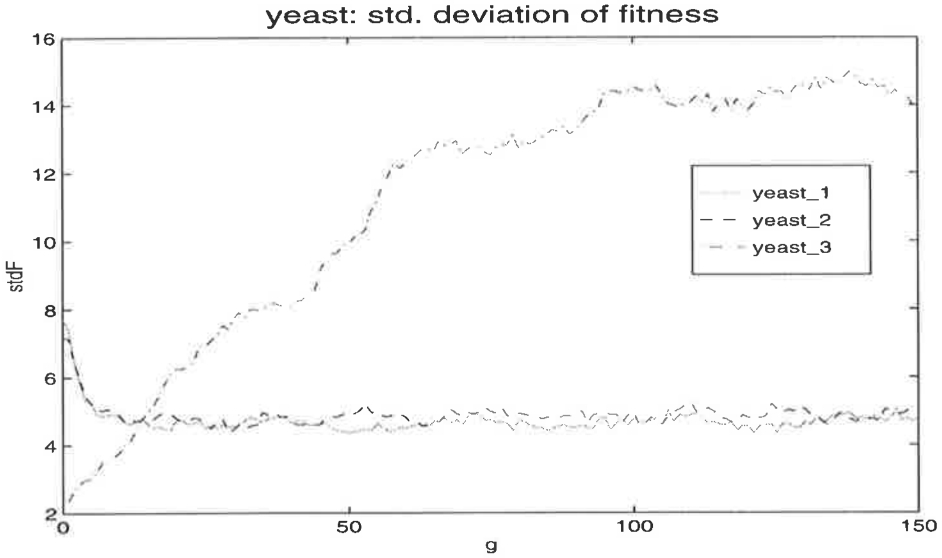


Figure 6.19: Standard deviation of fitness versus generation for all three permutations of **yeast**, averaged over 10 runs.

improve their performance by evolving appropriate features. When compared with existing techniques for classification, EPrep achieved error rates that were similar to the MLP and better than QUEST. Except for the kNN algorithm, the simple methods did not perform well on average. Although EPrep obtained the lowest estimate of test set error for more data sets than the MLP, the advantage was only statistically significant about a quarter of the time. The advantage of the MLP over EPrep was statistically significant more often. The MLP was generally more consistent from run to run than EPrep was, and achieved a test set error among the lowest three for more data sets.

EPrep yielded particularly good results for the **balance** data set, which is characterised by its discrete attributes that have a natural ordering, and are therefore treated as real-values. EPrep's performance on **satimage** was poor due to the large number of input measurements for that domain. Performance was also poor for the **yeast** problem, the reason most likely being an incompatibility between the structure in the data and the function set used. For each data set, it is possible that EPrep could achieve a lower error by better selection of the parameters. For the **vehicle** data set, it was impossible for EPrep to improve upon the maximum likelihood classifier, and the algorithm reverted to feature selection.

The results of EPrep and QUEST were compared for their interpretability. For some problems, such as **concentric**, EPrep had a clear advantage in understandability due to its ability to combine intuitively appropriate functions to form the decision boundaries. For other problems, EPrep's results were harder to read because the features can contain useless sub-expressions, the combination of multiple features can lead to discrimination in a non-intuitive manner, the way in which the classifier contributes to discrimination is unknown, and the features may require simplification by hand. Both methods sometimes resulted in solutions that were too large and complicated to be understood by a human. Both EPrep and QUEST were able to perform attribute selection by excluding input measurements not required for classification. There was general agreement between EPrep and QUEST on which measurements are required for discrimination for the **australian**, **segment** and **cmc** data sets. It was found for **titanic** that the relationships found in the data can vary with the permutation of the data. For the **monks** problems, QUEST sometimes split on variables that are known to be unimportant, while EPrep sometimes included the unimportant variables in useless expressions. Very different feature sets were obtained by EPrep on different runs and for different permutations of the data. The results of QUEST varied with permutation, but the variables used for the first few splits were generally the same. All these factors make interpretation of the data a complex matter which should be carried out with great care.

Of QUEST, the MLP and EPrep, QUEST was the most efficient algorithm, while EPrep was the most computationally expensive. If the local optimisation were switched off, EPrep could complete a run in about the same time as the MLP. The choice of whether to use EPrep depends on the trade-off between the expected benefits and the increased computational cost over other methods.

Regarding the EPrep algorithm, the RAT algorithm was able to effectively reduce the number of training samples processed on average for all data sets. In contrast, the self-adaptive operator probabilities and introns did not seem to work effectively. There was no consistent pattern to the behaviour of the operator probabilities from one run to the next, other than the seeming tendency towards the use of high-level crossover, which is most likely due to the inherent bias. The number of introns in the population was expected to increase with generations to protect highly-fit sub-sets of features, but tended to decrease with generations. An error in the program contributed to the lack of success of the intron accumulation. The local optimisation brought about a relatively small improvement in fitness on average, and does not appear to be worth the dramatic increase in computation time necessary. EPrep tended to use more features rather than fewer, which indicates that the classifier is being exploited by increasing the dimensionality of the feature space. The bloating phenomenon was not observed, which may be due to the explicit bias towards smaller individuals when breaking ties, or to the requirement that solutions generalise well, which implicitly biases the algorithm towards smaller pre-processors. An examination of the confusion matrices of the EPrep solutions indicates that there was confusion between classes that are intuitively similar, and that there was sometimes a bias towards classes with more samples in the database.

To conclude, EPrep was able to achieve similar classification accuracy to the MLP while providing information about the data.

Chapter 7

Conclusion

7.1 Introduction

The reader was provided with an introduction to pattern recognition in Chapter 2 and an overview of evolutionary computation in Chapter 3. The main hypotheses and research questions of the thesis were put forth in Chapter 4. The algorithm used to investigate the research questions was described in Chapter 5, and the experiments and results were presented in Chapter 6. This concluding chapter links the experimental results back to the original research questions and hypotheses, lists the implications for various fields of research, and contains recommendations for further investigation.

7.2 Conclusions about Research Questions and Hypotheses

The results of Chapter 6 are now discussed in the context of the research questions and hypotheses presented in Chapter 4. First, the research questions are discussed, then experimental support for the hypotheses is presented.

7.2.1 Question 4.1: Feasibility of Automatic Feature Extraction

The experiments of Chapter 6 have demonstrated that a search for a near-optimal generalised pre-processor is feasible for real-world supervised classification problems. Starting with simple classifiers that performed relatively poorly on their own, EPrep was able to improve their classification performance significantly by evolving features that were appropriate for the specific classifier. This improvement was statistically significant at the 99% level for 80% of the data sets used in the experiments.

The search was not confounded by a large number of classes, or by enumerated variables with large numbers of distinct values. Training time was linear in the number of training samples used, as is the case for most classification algorithms. The main sensitivity of performance to problem size was through the dimensionality of the data. This sensitivity was expected, because the size of the search space is a polynomial function of the data dimensionality, and the order of the polynomial increases exponentially with the maximum depth of the feature trees. EPrep achieved relatively poor error rates was when the function set did not contain the appropriate constituent functions for the problem, as was the case with the **yeast** data set.

7.2.2 Question 4.2: Automatic Feature Extraction for Knowledge Discovery

The generalised pre-processor method has advantages and disadvantages for knowledge discovery. On the positive side, the ability to combine arbitrary non-linear functions resulted in easily-understood solutions for **balance**, **concentric** and **titanic**. The arbitrary structure allows attribute selection to occur at the same time as feature extraction so that the most discriminatory input measurements can be identified. On the negative side, the presence of multiple features can make interpretation difficult, and their interaction with the classifier can blur the role of the features for discrimination. The pre-processors often contain expressions that must be simplified by hand, and superfluous expressions which do not contribute to fitness. The evolved features vary dramatically from run to run, which undermines confidence in interpretation of the results.

For the **balance**, **concentric** and **titanic** problems, useful relationships between the variables were extracted from the data. For instance, the user may not have previously known that third class passengers were discriminated against on the SS Titanic life-boats. Similarly, the automatic selection of attributes was revealing for the **australian** and **cmc** problems. The information file that came with the **australian** data set indicated that only five of the fourteen attributes were judged important by a stepwise regression procedure, whereas EPrep evolved pre-processors that used only two or three attributes, and disregarded four of the five supposedly-important attributes. This discrepancy indicates that it is dangerous to select the attributes based on only one method. For the **cmc** problem, EPrep revealed the interesting fact that religion was not related to the subject's choice of contraceptive method.

7.2.3 Hypothesis 4.1: Benefit of Appropriate Constituent Functions

The experiments have demonstrated the advantage of combining the appropriate constituent functions in particular situations. The use of the division function proved beneficial for the **balance** problem, so that the results obtained by EPrep were better than those of any other algorithm examined. The arithmetic functions used for **concentric1** resulted in a concise pre-processor for the problem containing 15 nodes, while the MLP which achieved the same accuracy had 202 adjustable weights, and the QUEST decision tree had 49 nodes. The use of the sine and cosine functions for the **spirals** problem gave EPrep an advantage over the MLP. For several of the real-world problems, such as **abalone**, **australian** and **titanic**, EPrep obtained accuracies similar to the MLP, but used a number of nodes an order-of-magnitude less than the number of weights used by the MLP.

In each of these examples, the combination of appropriate constituent functions allowed for a more realisable or, in the case of **balance**, more accurate classifier. When considering this comparison, however, it must be remembered that the classifier used in conjunction with the pre-processor also requires resources.

7.2.4 Hypothesis 4.2: Benefit of Appropriate Structure

The pre-processors shown in Chapter 6 do not have a regular structure like the MLP does. The removal of constraints on the pre-processor architecture has allowed EPrep to evolve more compact solutions than the MLP, as discussed in the previous sub-section. The pre-processors are not fully-connected from layer to layer, therefore not all of the input measurements are necessarily present in EPrep's solutions. Thus flexibility in structure allows EPrep to perform attribute selection while the MLP gives no immediately recognisable information as to which measurements are irrelevant for classification.

7.2.5 Hypothesis 4.3: Advantage of the Generalised Pre-Processor

From the comparative experiments, it can be concluded that there are problems for which the generalised pre-processor method is preferable to the other classification methods examined. In particular, EPrep was an advantageous method to use for the **balance** problem. The EPrep algorithm had the best peak performance in that it achieved the lowest test set error for more problems than the other classifiers examined. However, not all of these results were statistically significant. In terms of those results that were statistically significant at the 95% level, the MLP was the best algorithm, with EPrep a close second, and QUEST a more distant third. Similarly in terms of reliability, the MLP was the most reliable algorithm in that it was one of the three best methods for the most problems, EPrep was second, and QUEST was third. The results of the MLP were more reliable from run to run than those of EPrep.

Although the MLP algorithm was on average the most accurate and reliable, it offers no interpretable information for knowledge discovery. EPrep and QUEST, however, can perform attribute selection and reveal relationships in the data. For the **concentric** data set, EPrep had a more understandable solution than QUEST due to its ability to combine intuitively appropriate functions to form the decision boundaries. In other cases, the difficulties in interpreting EPrep's features made the QUEST trees more understandable. Both algorithms used variables that were known to be unimportant for the **monks** problems, but this only seemed to interfere with the classification accuracy of the QUEST method.

There was some consensus between EPrep and QUEST on the importance of certain attributes for the **australian**, **segment** and **cmc** data sets. The relationships found in the data for **titanic** were also in general agreement for the two algorithms. For the **smoking** data set, the QUEST tree had only a single node whereas the over-fitted EPrep pre-processor was quite large and complicated. This exhibits the danger of using a single method for knowledge discovery. The results of QUEST were more robust to permutation of the data, and did not suffer from the variability over runs that EPrep's pre-processors did.

In terms of computational complexity, QUEST was the simplest algorithm and EPrep was the most intensive. With local optimisation switched off, however, EPrep's computation time is approximately the same as that of the MLP.

In summary, the use of a generalised pre-processor did not bring about significant improvements in classification accuracy over other methods for real-world problems. The generalised pre-processors were, however, more practically realisable than the MLP in the sense that they generally required fewer nodes and fewer input measurements. An additional feature of the method is that the economical pre-processors and classifiers can be easily implemented on a micro-processor. The function set could even be constrained to those functions available on the micro-processor. In some instances, the pre-processors were more informative about the data than the decision tree method. Since neither EPrep nor QUEST are reliable enough to use in isolation for knowledge discovery, EPrep can be used as an additional source of information.

7.2.6 Hypothesis 4.4: Advantage of Population-Based Search

The performance curves in Appendix E show that, in many cases, the average size of solutions in the population did decrease with generations. In particular, a size decrease was observed for all of the synthetic problems. The most poignant example of an exponential decrease was for the **vehicle** problem, which displayed a very rapid drop in average pre-processor size. In several of the cases for which an initial size decrease was observed, the size later increased. The two explanations for this are over-fitting and bloating. In most cases where the average pre-processor size increased, the training error continued to improve as well. Hence the cause for the size increase was over-fitting rather than bloating.

7.3 Conclusions about the Performance of EPrep

Additional conclusions can be made about the performance of the evolutionary pre-processor algorithm itself. The rational-allocation-of-trials algorithm worked well at decreasing the average number of samples processed per individual. This is a promising addition to the standard genetic or evolutionary algorithm. The use of many genetic operators successfully maintained diversity for the duration of each run. The self-adaptive operator probabilities, however, did not adapt with any pattern, and the mechanism seemed to be of little use. The introns tended to trickle away rather than building up in number to protect co-adapted sets of features. A similar lack of improvement through the use of inversion has been reported in (Goldberg, 1989, pp. 166-170). The average improvement in classification rate brought about by local optimisation was so slight as to not be worth the enormous increase in computation time required.

The bloating phenomenon did not often occur: in those instances where the average size of individuals in the population increased with generations, the fitness was also changing. This indicates that the individuals were increasing in size due to over-fitting rather than bloating. The absence of bloating was expected for several reasons:

- Those offspring having the same fitness as their parent but containing more nodes were given a lower rank in the population due to a preference towards parsimony during sorting.
- According to Occam's principle, more parsimonious solutions should generalise better. Since the time-varying fitness function resulted in a fitness criterion of generalisation, there was an intrinsic bias towards smaller pre-processors.
- Sub-tree crossover was not used. Therefore trees could only increase in size by application of the grow mutation operator. This operator was not applied frequently enough to cause a radical growth in average feature size.

EPrep tended to generate more features as generations progressed, but nevertheless most of the best evolved pre-processors reduced the dimensionality of the data from input to output. The generalisation performance of the evolved features used with the simple classifiers was generally very good, and the run with the minimum validation set error was usually the run with the minimum test set error. However, the pre-processors themselves varied significantly from run to run. It was sometimes difficult to choose acceptable parameters for EPrep, as was the case for the **yeast** problem.

If one of the simple classifiers used by EPrep was already achieving an error rate close to the Bayes limit for the problem, there was little scope for improvement and EPrep reverted to feature selection. Such was the case for the **vehicle** data set. From the confusion matrices of the best solutions generated by EPrep, it could be seen that intuitively similar classes were often confused. For problems with disproportionate class representations in the data set, EPrep tended to favour the more frequent classes in its decisions. This problem can be controlled through the use of cost matrices.

7.4 Conclusions about the Research Topic

In Chapter 1, the following question was stated to summarise the topic of this thesis:

> *How effective is a generalised pre-processor at extracting features from real data when compared with existing automatic feature extraction methods?*

The answer revealed by the examinations of this thesis depends on the criterion employed. In terms of classification accuracy, the GPP approach was effective in that it was able to improve the performance of simple classifiers. For the real-world data sets, however, the GPP approach did not result in significantly better classification rates than the MLP. In terms of utility for knowledge discovery, the GPP method resulted in more informative features than the MLP. The GPP approach had advantages over decision trees due to its versatility in combining different functions. Nevertheless, the decision trees were generally more reliable. In terms of computational complexity, the search for the best GPP was relatively expensive, but the resulting classification systems were relatively economical, both in computational resources and in the number of input measurements required.

Whatever the criterion used to assess the performance of the GPP search method, the effectiveness of the algorithm broke down when the dimensionality of the data was large.

7.5 Implications for Theory

The findings of this thesis have implications for feature extraction in general. The fixed structure and limited but smooth constituent functions used in the multi-layer perceptron are sufficient to extract features from the real-world data sets examined. Nevertheless, the fact that the combination of different non-linear functions can achieve better results for some synthetic problems suggests that the generalised pre-processor methodology would be advantageous in certain real scenarios. By removing constraints on the structure and content of a feature extractor, the system can be realised more economically. When problems are scaled up, this economy may mean the difference between a feasible and an infeasible real-time system.

The experimental procedures and findings have general implications for the field of supervised learning. The use of multiple permutations of each data set has shown the danger of drawing conclusions from a single permutation of the data. In comparing different classification techniques, relatively few of the differences in error rate were statistically significant. The interpretation of the models obtained by the classification methods also varied with permutation of the data. All of these findings indicate that machine learning algorithms cannot be compared *prima facie*, and that their use for knowledge discovery must be performed with great care. Amidst the recent excitement about knowledge discovery in databases for practical applications, such caution is scarce. It is recommended that several different methods be used for knowledge discovery, and their results cross-referenced. The evolutionary pre-processor is one such method that can be used in conjunction with other techniques.

The implications for the No Free Lunch theorem are that some general algorithms are more tuned to the problems people are interested in than others. For instance, the MLP ranked near the top the most often, and some of the simpler methods such as the GLIM performed quite badly across the problem set. The main result of the NFL theorem, that all algorithms perform equally on average, was vindicated in that no algorithm was the best for all problems examined.

There were important findings for the field of genetic programming. The rational-allocation-of-trials algorithm was quite effective at reducing the number of fitness evaluations, and should find more widespread use amongst the GP community. The method used for self-adaptation of operator probabilities, however, was not effective and needs improvement. The use of the simplex method and hill-climbing for local optimisation did not seem to be worth the overhead on average. It is difficult to assess the value of optimisation overall, however, because it may have brought about large improvements for a few key individuals and thus improved the quality of the final solutions. The fact that the co-operation of high-level introns and the inversion operator did not work to protect co-adapted features from destructive crossover may be an indication that there was no schema accumulation occurring. This sustains the argument in (Angeline, 1997*b*) that for many problems, crossover is a form of macro-mutation.

The final contribution of this thesis to scientific research as a whole is to question popular assumptions. Sometimes gains can be made by framing a problem in a more general manner.

7.6 Implications for Further Research

The two most severe limitations of the EPrep algorithm are its computational complexity and its sensitivity to the dimensionality of the input data. The most interesting avenue of further research would be to extend EPrep to work with high-dimensional data such as images and time-series. This could be achieved by adding loops and memory to the pre-processors so that a relatively small pre-processor could be iteratively applied to the data. Similar work has already been conducted by (Koza, 1994*b*) and (Teller and Veloso, 1996). The increase in data size would bring about a significant increase in computation time, which would have to be somehow circumvented. Methods would also have to be developed to cope with sparse sampling of the very-high dimensional data.

Further research is required into mechanisms for self-adaptation of parameters in genetic programming. Cross-fertilisation with methods for partial credit assignment from the artificial intelligence community may produce effective algorithms for adapting operator probabilities. To reduce the computational overhead of local optimisation, algorithms could be developed to apply different amounts of computational resources to different individuals. For instance, individuals whose ancestors have been optimised and that are similar to those ancestors may receive relatively few optimisation trials in future, under the assumption that they are already relatively well optimised.

The field of knowledge discovery is now becoming important, but the main drawback of EPrep for this use is the high variability in the results. Future research could endeavour to constrain the solution representation so as to produce more consistent results from run to run. Methods could be developed to remove superfluous expressions from the features. The performance of EPrep could also be improved by investigating what set of complementary constituent functions is the most appropriate for a broad range of problems.

Appendix A

Instructions for Use of CD-Rom

Due to the enormous amount of data produced by EPrep, a CD-rom disk has been attached to the back of this thesis. The CD-rom can be used with a Unix workstation, or with a PC under Windows 95 or Windows NT. The CD-rom contains the following directories:

***results*:** HTML reports, data and figures generated by EPrep in the experiments of Chapter 6.

***data*:** data files and types files used with EPrep in the experiments of Chapter 6.

***parameters*:** parameter files used with EPrep in the experiments of Chapter 6.

***eprep*:** executable files of EPrep and associated utilities, for use with DEC and Sun computers. Note that the EPrep GUI is not included.

The item of most interest to the reader is the collection of reports output by EPrep. These reports can be viewed with a web browser such as Netscape or Microsoft Internet Explorer through the top-level HTML page:

***cd-rom*/index.html**

where *cd-rom* is the path of your CD-rom drive. Simply insert the CD-rom disk into the drive, and select the top-level HTML file through the internet browser.

Appendix B

Implementation Details of EPrep

This appendix contains some of the history and details of the evolutionary pre-processor software implementation, and motivates the major design choices.

B.1 Development History

The EPrep algorithm and software described in this thesis are the latest version, EPrep 3.0. The previous version of EPrep were significantly different: due to poor maintainability of the software and algorithmic shortcomings, the software was entirely re-written from scratch.

Before version 3.0, the implementation of EPrep was a mixture of three software components: proprietary code, the XView Graphical-User Interface (GUI) library (Microsystems, 1991), and Matthew's GALib genetic algorithm library (Wall, 1996). The software was written in C and C++ for Unix and Linux workstations. Over 21,000 lines of code (LOC) were written for version 2.5 of the software (this does not include the GALib or XView libraries). Version 3.0 of EPrep was written in C++ (31,000 LOC) with a Java GUI (6,700 LOC) for a Unix workstation.

Although the total abandonment of the previous implementation of EPrep may seem like time wasted, many invaluable lessons were learned about the implementation and algorithmic details, which resulted in a final algorithm of higher quality.

B.2 Design Choices

Some of the global design choices involved with design of the latest version of the software are discussed here. These choices were motivated by the following difficulties encountered with previous versions.

B.2.1 Maintainability

Although GALib is a reusable object-oriented library, it was not flexible enough to neatly accommodate the structures and algorithms of EPrep. The GUI library XView, like many GUI libraries, was very messy to integrate with proprietary code. Consequently the previous version of EPrep was expensive to maintain as a piece of software. The complicated modifications to be made for the next version would have been extremely difficult to implement by modifying the existing GALib, and would have required the re-writing of many components.

As an alternative to the band-aid approach, the GP kernel was re-written resulting in *Jamie's Genetic Program* (JGP), a generally re-usable C++ object-oriented strongly-typed genetic programming library. JGP was designed specifically to accommodate the modifications to GP that were required by EPrep 3.0.

B.2.2 Portability

The XView library only worked under Solaris and Linux operating systems, so the GUI could not be used on other platforms. This restricted future availability to any parties interested in the use of EPrep. A version of the software without the GUI is available, and can be configured through text files, so one may wonder if a GUI is necessary to perform the experiments at all. The user interface is useful for two reasons:

- the user can see the results plotted versus generation as they occur, and examine the contents of the population at arbitrary generations. This is useful for understanding how the algorithm operates, and for heuristic selection of learning parameters.
- if an external party wished to use the software, the GUI is invaluable for user-friendliness and their understanding of the algorithm.

The solution to the problem of GUI portability has two parts. The first is to implement an interface class to communicate between the GUI and EPrep, as shown in Figure B.1. This has the attractive quality that the GUI can be replaced by a command-line interpreter or a data file. The second part is to use a platform-independent GUI library, such as Java (Flanagan, 1997) or V (Wampler, 1998). This cuts down on programming if multiple platforms need to be supported, but technically the GUI could be re-written using any GUI library for the platform in mind.

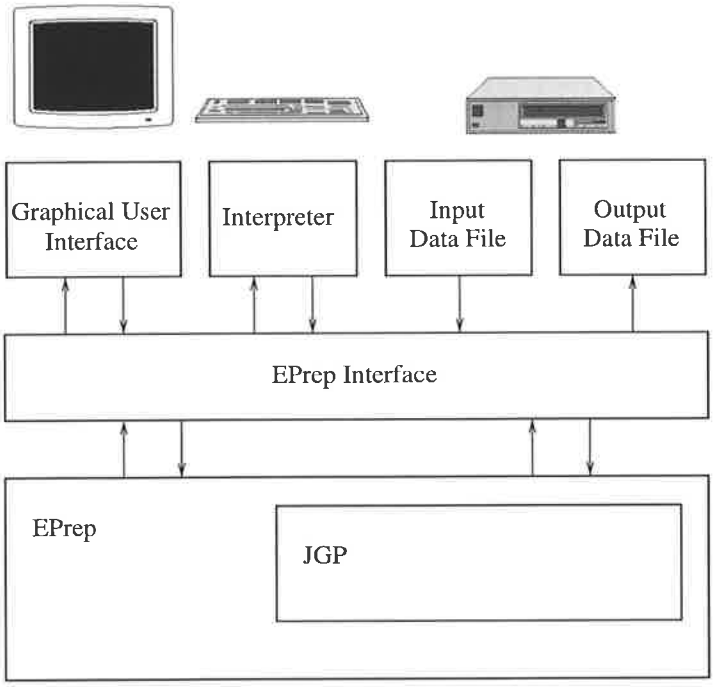


Figure B.1: The Large-Scale Architecture of EPrep, showing the interface class.

It was preferable to implement the interface class regardless of the choice of GUI library, because this insulates[1] the GUI writer from the details of the EPrep software.

The Java Development Kit version 1.1 (Microsystems, 1998) was chosen to implement the GUI due to its continued support, alleged platform independence, and easy use of call-back functions and threads. The resulting combination of C++ and Java code required the use

[1] *Insulation* is the practice of making implementation details programmatically inaccessible to the library user (Lakos, 1996).

of *native methods*, which are Java methods written in C (Stearns, 1997). To date there have been problems with using native methods on DEC machines, and the necessary modifications have not been made to the C++ code to enable use on a PC, so the GUI version of EPrep currently works on Sun workstations only.

B.2.3 Speed

The issue of speed is very important, since the EPrep algorithm does not scale well with problem size and some simulations using the previous version have taken weeks to compute. While the objective function is the dominating time factor in most evolutionary algorithms, the evaluation of a tree expression can also be extremely time consuming: the number of nodes in a tree is exponential in the depth, and can become quite large. The previous implementation stored trees as records connected with pointers, which is not the most efficient implementation for evaluating individuals. There were many inefficiencies in GALib as well, since generality and reusability generally incurs a run-time overhead in pointer dereferencing and virtual table look-ups.

A comparison of several GP implementations in C++ by (Keith and Martin, 1994) revealed that a linear prefix and jump table approach is overall the most efficient implementation. This approach stores the expression trees in prefix notation in a linear array. Each array element contains an index into a jump table where the node information is stored. Execution of an individual proceeds by initialising a program counter at the start of the array, and iterating a lookup-evaluate cycle until no more arguments are expected. Only two bytes are required per node. Although the tree of pointers method is the fastest method in terms of number of instructions required for evaluation, it has a much larger memory overhead which, in the long run, can slow the system down due to disk swapping and restrict the largest possible population size.

Appendix C

Description of Data Sets

This Appendix contains a description of each of the 15 public-domain data sets used in the experiments of Chapter 6. Each data set has a sub-section with a brief description of the classification problem, and a table containing the details of the data set. The fields of the tables are:

dimensions: number of input attributes from which to predict the class label.

classes: number of distinct classes.

samples: number of examples in the database.

preparation: steps carried out on the data to prepare it for use with EPrep.

partition: number of samples in the training, validation and test sets; $n_{tr} - n_{val} - n_{tst}$.

source: whence the data came. All data sets used (except for **cmc**) come from the following repositories:

cmu	CMU Neural Network Benchmark Database (White and Fahlman, 1993)
delve	Data for Evaluating Learning in Valid Experiments (Rasmussen *et al.*, 1996)
elena	Enhanced Learning for Evolutive Neural Architecture (Aviles-Cruz *et al.*, 1995)
statlib	StatLib: a system for distributing statistical software, datasets, and information by electronic mail, FTP and WWW (Meyer, 1996)
statlog	Project StatLog (ESPRIT, 1995)
uci	UCI Repository of Machine Learning (Merz and Murphy, 1996)

default error: the percentage of misclassifications made on the whole data set by always guessing the most frequent class.

input attributes: the name, type and range of values for each input attribute from the database. Possible types are real, enum(erated) and bool(ean). Note that some attributes are discrete but are still considered to be real-valued rather than enumerated because the ordering over the values makes sense; *eg:* number of children, age. Some enumerated values only have two possible values and could be considered boolean: this is an arbitrary choice.

missing values: describes how missing values were handled by the original collectors of the data.

class distribution: the name of each class, along with the number and proportion of samples from the database belonging to that class.

C.1 abalone

Predicting the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope – a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the age. Further information, such as weather patterns and location (hence food availability) may be required to solve the problem.

From the original data, examples with missing values were removed (the majority having the predicted value missing), and the ranges of the continuous values have been scaled by dividing by 200.

The details of the **abalone** data set are shown in Table C.1. Note that some of the classes contain only one or two samples.

Table C.1: Details of the **abalone** data set.

<table>
<tr><td>dimensions</td><td>8</td></tr>
<tr><td>classes</td><td>29</td></tr>
<tr><td>samples</td><td>4177</td></tr>
<tr><td>preparation</td><td>For attribute 1, replaced M with 1, F with 2, and I with 3. Subtracted 1 from class labels.</td></tr>
<tr><td>partition</td><td>2088-1044-1045</td></tr>
<tr><td>source</td><td>UCI</td></tr>
<tr><td>default error</td><td>83.51%</td></tr>
<tr><td>input attributes</td><td>
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>Sex</td><td>enum</td><td>male, female, infant</td></tr>
<tr><td>2</td><td>Length</td><td>real</td><td>0.075, . . . , 0.815</td></tr>
<tr><td>3</td><td>Diameter</td><td>real</td><td>0.055, . . . , 0.650</td></tr>
<tr><td>4</td><td>Height</td><td>real</td><td>0.000, . . . , 1.130</td></tr>
<tr><td>5</td><td>Whole weight</td><td>real</td><td>0.002, . . . , 2.826</td></tr>
<tr><td>6</td><td>Shucked weight</td><td>real</td><td>0.001, . . . , 1.488</td></tr>
<tr><td>7</td><td>Viscera weight</td><td>real</td><td>0.001, . . . , 0.760</td></tr>
<tr><td>8</td><td>Shell weight</td><td>real</td><td>0.002, . . . , 1.005</td></tr>
</table>
</td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td>
<table>
<tr><th>class (age)</th><th>samples</th><th>proportion(%)</th></tr>
<tr><td>1</td><td>1</td><td>0.0239</td></tr>
<tr><td>2</td><td>1</td><td>0.0239</td></tr>
<tr><td>3</td><td>15</td><td>0.3591</td></tr>
<tr><td>4</td><td>57</td><td>1.3646</td></tr>
<tr><td>5</td><td>115</td><td>2.7532</td></tr>
<tr><td>6</td><td>259</td><td>6.2006</td></tr>
<tr><td>7</td><td>391</td><td>9.3608</td></tr>
<tr><td>8</td><td>568</td><td>13.5983</td></tr>
<tr><td>9</td><td>689</td><td>16.4951</td></tr>
<tr><td>10</td><td>634</td><td>15.1784</td></tr>
<tr><td>11</td><td>487</td><td>11.6591</td></tr>
<tr><td>12</td><td>267</td><td>6.3921</td></tr>
<tr><td>13</td><td>203</td><td>4.8599</td></tr>
<tr><td>14</td><td>126</td><td>3.0165</td></tr>
<tr><td>15</td><td>103</td><td>2.4659</td></tr>
<tr><td>16</td><td>67</td><td>1.6040</td></tr>
<tr><td>17</td><td>58</td><td>1.3886</td></tr>
<tr><td>18</td><td>42</td><td>1.0055</td></tr>
<tr><td>19</td><td>32</td><td>0.7661</td></tr>
<tr><td>20</td><td>26</td><td>0.6225</td></tr>
<tr><td>21</td><td>14</td><td>0.3352</td></tr>
<tr><td>22</td><td>6</td><td>0.1436</td></tr>
<tr><td>23</td><td>9</td><td>0.2155</td></tr>
<tr><td>24</td><td>2</td><td>0.0479</td></tr>
<tr><td>25</td><td>1</td><td>0.0239</td></tr>
<tr><td>26</td><td>1</td><td>0.0239</td></tr>
<tr><td>27</td><td>2</td><td>0.0479</td></tr>
<tr><td>29</td><td>1</td><td>0.0239</td></tr>
<tr><td>total</td><td>4177</td><td>100.00</td></tr>
</table>
</td></tr>
</table>

C.2 australian credit

Classify credit card applications as successful or unsuccessful. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data. This dataset is interesting because there is a good mix of attributes – continuous, nominal with small numbers of values, and nominal with larger numbers of values. There were originally a few missing values, but these have all been replaced by the overall median.

A stepwise regression procedure strongly suggests that only attributes A5, A8, A9, A13 and A14 are relevant. Improved results are often obtained if only these five attributes are used.

Table C.2: Details of the **australian** data set.

<table>
<tr><td>dimensions</td><td>14</td></tr>
<tr><td>classes</td><td>2</td></tr>
<tr><td>samples</td><td>690</td></tr>
<tr><td>preparation</td><td>none</td></tr>
<tr><td>partition</td><td>345-172-173</td></tr>
<tr><td>source</td><td>statlog</td></tr>
<tr><td>default error</td><td>44.5%</td></tr>
<tr><td>input attributes</td><td>descriptions are confidential
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>A1</td><td>enum</td><td>a,b</td></tr>
<tr><td>2</td><td>A2</td><td>real</td><td>13.75,...,80.25</td></tr>
<tr><td>3</td><td>A3</td><td>real</td><td>0,...,28.00</td></tr>
<tr><td>4</td><td>A4</td><td>enum</td><td>p,g,gg</td></tr>
<tr><td>5</td><td>A5</td><td>enum</td><td>ff,d,i,k,j,aa,m,c,w,e,q,r,cc,x</td></tr>
<tr><td>6</td><td>A6</td><td>enum</td><td>ff,dd,j,bb,v,n,o,h,z</td></tr>
<tr><td>7</td><td>A7</td><td>real</td><td>0,...,28.5</td></tr>
<tr><td>8</td><td>A8</td><td>boolean</td><td>true,false</td></tr>
<tr><td>9</td><td>A9</td><td>boolean</td><td>true,false</td></tr>
<tr><td>10</td><td>A10</td><td>real</td><td>0,...,67.00</td></tr>
<tr><td>11</td><td>A11</td><td>boolean</td><td>true,false</td></tr>
<tr><td>12</td><td>A12</td><td>enum</td><td>s, g, p</td></tr>
<tr><td>13</td><td>A13</td><td>real</td><td>0,...,2000</td></tr>
<tr><td>14</td><td>A14</td><td>real</td><td>1,...,100001</td></tr>
</table></td></tr>
<tr><td>missing values</td><td>37 cases (5%) of the original data had one or more missing values. The missing values from particular attributes were:
<table>
<tr><td>A1</td><td>A2</td><td>A4</td><td>A5</td><td>A6</td><td>A7</td><td>A14</td></tr>
<tr><td>12</td><td>12</td><td>6</td><td>6</td><td>9</td><td>9</td><td>13</td></tr>
</table>
These were replaced by the mode of the attribute (categorical) mean of the attribute (continuous).</td></tr>
<tr><td>class distribution</td><td>
<table>
<tr><th>class</th><th>samples</th><th>proportion(%)</th></tr>
<tr><td>successful</td><td>307</td><td>44.5</td></tr>
<tr><td>unsuccessful</td><td>383</td><td>55.5</td></tr>
<tr><td>total</td><td></td><td>100.00</td></tr>
</table></td></tr>
</table>

C.3 balance

This data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance. The correct way to find the class is the greater of (left-distance × left-weight) and (right-distance × right-weight). If they are equal, it is balanced.

Table C.3: Details of the **balance** data set.

<table>
<tr><td>dimensions</td><td>4</td></tr>
<tr><td>classes</td><td>3</td></tr>
<tr><td>samples</td><td>625</td></tr>
<tr><td>preparation</td><td>moved class labels to last column, replaced letters: L with 0, B with 1, R with 2</td></tr>
<tr><td>partition</td><td>312-156-157</td></tr>
<tr><td>source</td><td>UCI</td></tr>
<tr><td>default error</td><td>53.92%</td></tr>
<tr><td>input attributes</td><td>
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>Left-Weight</td><td>real</td><td>1, 2, 3, 4, 5</td></tr>
<tr><td>2</td><td>Left-Distance</td><td>real</td><td>1, 2, 3, 4, 5</td></tr>
<tr><td>3</td><td>Right-Weight</td><td>real</td><td>1, 2, 3, 4, 5</td></tr>
<tr><td>4</td><td>Right-Distance</td><td>real</td><td>1, 2, 3, 4, 5</td></tr>
</table>
</td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td>
<table>
<tr><th>class</th><th>samples</th><th>proportion (%)</th></tr>
<tr><td>left</td><td>288</td><td>46.08</td></tr>
<tr><td>balanced</td><td>49</td><td>7.84</td></tr>
<tr><td>right</td><td>288</td><td>46.08</td></tr>
<tr><td>total</td><td>625</td><td>100.00</td></tr>
</table>
</td></tr>
</table>

C.4 concentric

A synthetic data set consisting of 2-dimensional points distributed in a circle of radius 0.3 centred on (0.5, 0.5) (class 0), and in an immediately-surrounding concentric annulus with external radius 0.5 (class 1). The data are plotted in Figure C.1, from (Aviles-Cruz *et al.*, 1995).

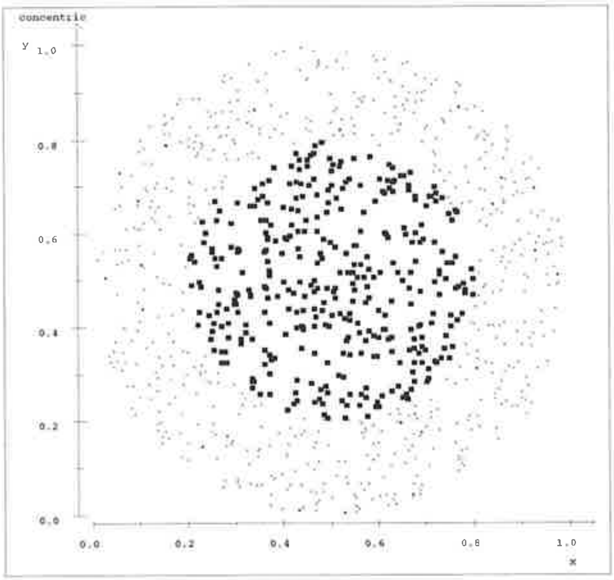


Figure C.1: Plot of **concentric** data set.

Table C.4: Details of the **concentric** data set.

<table>
<tr><td>dimensions</td><td>2</td></tr>
<tr><td>classes</td><td>2</td></tr>
<tr><td>samples</td><td>2500</td></tr>
<tr><td>preparation</td><td>nothing.</td></tr>
<tr><td>partition</td><td>1250-625-625</td></tr>
<tr><td>source</td><td>elena</td></tr>
<tr><td>default error</td><td>36.84%</td></tr>
<tr><td>input attributes</td><td><table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>x</td><td>real</td><td>0, . . . , 1</td></tr>
<tr><td>2</td><td>y</td><td>real</td><td>0, . . . , 1</td></tr>
</table></td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td><table>
<tr><th>class</th><th>samples</th><th>proportion(%)</th></tr>
<tr><td>circle</td><td>921</td><td>36.84</td></tr>
<tr><td>annulus</td><td>1579</td><td>63.16</td></tr>
<tr><td>total</td><td>2500</td><td>100.00</td></tr>
</table></td></tr>
</table>

C.5 contraceptive method choice (cmc)

This dataset is a subset of the 1987 National Indonesia Contraceptive Prevalence Survey. The samples are married women who were either not pregnant or do not know if they were at the time of interview. The problem is to predict the current contraceptive method choice (no use, long-term methods, or short-term methods) of a woman based on her demographic and socio-economic characteristics.

Table C.5: Details of the **cmc** data set.

<table>
<tr><td>dimensions</td><td>9</td></tr>
<tr><td>classes</td><td>3</td></tr>
<tr><td>samples</td><td>1473</td></tr>
<tr><td>preparation</td><td>Subtracted 1 from class labels.</td></tr>
<tr><td>partition</td><td>736-368-369</td></tr>
<tr><td>source</td><td>Wei-Yin Loh, http://www.stat.wisc.edu/~loh/loh.html</td></tr>
<tr><td>default error</td><td>57.30%</td></tr>
<tr><td>input attributes</td><td>
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>Wife's age</td><td>real</td><td>16, . . . , 49</td></tr>
<tr><td>2</td><td>Wife's education</td><td>real</td><td>1, 2, 3, 4</td></tr>
<tr><td>3</td><td>Husband's education</td><td>real</td><td>1, 2, 3, 4</td></tr>
<tr><td>4</td><td>Number of children ever born</td><td>real</td><td>0, . . . , 16</td></tr>
<tr><td>5</td><td>Wife's religion</td><td>boolean</td><td>non-islam, islam</td></tr>
<tr><td>6</td><td>Wife is now working?</td><td>boolean</td><td>yes, no</td></tr>
<tr><td>7</td><td>Husband's occupation</td><td>enum</td><td>1, 2, 3, 4</td></tr>
<tr><td>8</td><td>Standard-of-living index</td><td>real</td><td>1, 2, 3, 4</td></tr>
<tr><td>9</td><td>Media exposure</td><td>boolean</td><td>good, not good</td></tr>
</table>
</td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td>
<table>
<tr><th>class</th><th>samples</th><th>proportion(%)</th></tr>
<tr><td>No-use</td><td>629</td><td>42.70</td></tr>
<tr><td>Long-term</td><td>333</td><td>22.61</td></tr>
<tr><td>Short-term</td><td>511</td><td>34.69</td></tr>
<tr><td>total</td><td>1473</td><td>100.00</td></tr>
</table>
</td></tr>
</table>

C.6 diabetes

The task is to determine whether the patient shows signs of diabetes according to World Health Organization criteria (*ie:* if the 2 hour post-load plasma glucose was at least 200 mg/dl at any survey examination, or if found during routine medical care). The population under study is the tribe of Pima Indians, living near Phoenix, Arizona, USA.

Table C.6: Details of the **diabetes** data set.

<table>
<tr><td>dimensions</td><td>8</td></tr>
<tr><td>classes</td><td>2</td></tr>
<tr><td>samples</td><td>768</td></tr>
<tr><td>preparation</td><td>removed commas</td></tr>
<tr><td>partition</td><td>384-192-192</td></tr>
<tr><td>source</td><td>UCI</td></tr>
<tr><td>default error</td><td>35%</td></tr>
<tr><td>input attributes</td><td>
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>Number of times pregnant</td><td>real</td><td>0-17</td></tr>
<tr><td>2</td><td>Plasma glucose concentration after 2 hours</td><td>real</td><td>0-199</td></tr>
<tr><td>3</td><td>Diastolic blood pressure (mm Hg)</td><td>real</td><td>0-122</td></tr>
<tr><td>4</td><td>Triceps skin fold thickness (mm)</td><td>real</td><td>0-99</td></tr>
<tr><td>5</td><td>2-Hour serum insulin (mu U/ml)</td><td>real</td><td>0-846</td></tr>
<tr><td>6</td><td>Body mass index (weight in kg/(height in m)^2)</td><td>real</td><td>$0 \ldots 67.1$</td></tr>
<tr><td>7</td><td>Diabetes pedigree function</td><td>real</td><td>$0.078 \ldots 2.42$</td></tr>
<tr><td>8</td><td>Age (years)</td><td>real</td><td>21-81</td></tr>
</table>
</td></tr>
<tr><td>missing values</td><td>Although there are no missing values in this data set according to its documentation, there are several senseless 0 values. These most probably indicate missing data. Nevertheless, we handle this data as if it was real, thereby introducing some errors (or noise, if you want) into the data set.</td></tr>
<tr><td>class distribution</td><td>
<table>
<tr><th>class</th><th>samples</th><th>proportion (%)</th></tr>
<tr><td>no diabetes</td><td>500</td><td>65</td></tr>
<tr><td>diabetes</td><td>268</td><td>35</td></tr>
<tr><td>total</td><td>768</td><td>100.00</td></tr>
</table>
</td></tr>
</table>

C.7 german credit

Credit card approval problem from a German bank. Classify applicants as successful or unsuccessful.

Table C.7: Details of the **german** data set.

<table>
<tr><td>dimensions</td><td>20</td></tr>
<tr><td>classes</td><td>2</td></tr>
<tr><td>samples</td><td>1000</td></tr>
<tr><td>preparation</td><td>Replaced AxXX with XX; subtracted 1 from class labels.</td></tr>
<tr><td>partition</td><td>500-250-250</td></tr>
<tr><td>source</td><td>statlog</td></tr>
<tr><td>default error</td><td>30%</td></tr>
<tr><td>input attributes</td><td>
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>Status of existing checking account</td><td>enum</td><td>A11 : ... &lt; 0 DM
A12 : 0 ≤ ... &lt; 200 DM
A13 : ... ≥ 200 DM salary assignments for at least 1 year
A14 : no checking account</td></tr>
<tr><td>2</td><td>Duration in month</td><td>real</td><td>4, ..., 72</td></tr>
<tr><td>3</td><td>Credit history</td><td>enum</td><td>A30 : no credits taken, all credits paid back duly
A31 : all credits at this bank paid back duly
A32 : existing credits paid back duly till now
A33 : delay in paying off in the past
A34 : critical account, other credits existing (not at this bank)</td></tr>
<tr><td>4</td><td>Purpose</td><td>enum</td><td>A40 : car (new)
A41 : car (used)
A42 : furniture/equipment
A43 : radio/television
A44 : domestic appliances
A45 : repairs
A46 : education
A47 : (vacation - does not exist?)
A48 : retraining
A49 : business
A410 : others</td></tr>
<tr><td>5</td><td>Credit amount</td><td>real</td><td>250, ..., 18424</td></tr>
<tr><td>6</td><td>Savings account/bonds</td><td>enum</td><td>A61 : ... &lt; 100 DM
A62 : 100 ≤ ... &lt; 500 DM
A63 : 500 ≤ ... &lt; 1000 DM
A64 : ... ≥ 1000 DM
A65 : unknown/ no savings account</td></tr>
<tr><td>7</td><td>Present employment since</td><td>enum</td><td>A71 : unemployed
A72 : ... &lt; 1 year
A73 : 1 ≤ ... &lt; 4 years
A74 : 4 ≤ ... &lt; 7 years
A75 : ... ≥ 7 years</td></tr>
<tr><td>8</td><td>Installment rate in percentage of disposable income</td><td>real</td><td>1, ..., 4</td></tr>
<tr><td>9</td><td>Personal status and sex</td><td>enum</td><td>A91 : male : divorced/separated
A92 : female : divorced/separated/married
A93 : male : single
A94 : male : married/widowed
A95 : female : single</td></tr>
<tr><td>10</td><td>Other debtors / guarantors</td><td>enum</td><td>A101 : none
A102 : co-applicant
A103 : guarantor</td></tr>
<tr><td>11</td><td>Present residence since</td><td>real</td><td>1, ..., 4</td></tr>
<tr><td>12</td><td>Property</td><td>enum</td><td>A121 : real estate
A122 : if not A121 : building society savings agreement/ life insurance
A123 : if not A121/A122 : car or other, not in attribute 6
A124 : unknown / no property</td></tr>
<tr><td>13</td><td>Age in years</td><td>real</td><td>19, ..., 75</td></tr>
<tr><td>14</td><td>Other installment plans</td><td>enum</td><td>A141 : bank
A142 : stores
A143 : none</td></tr>
<tr><td>15</td><td>Housing</td><td>enum</td><td>A151 : rent
A152 : own
A153 : for free</td></tr>
<tr><td>16</td><td>Number of existing credits at this bank</td><td>real</td><td>1, ..., 4</td></tr>
<tr><td>17</td><td>Job</td><td>enum</td><td>A171 : unemployed/ unskilled - non-resident
A172 : unskilled - resident
A173 : skilled employee / official
A174 : management/ self-employed/ highly qualified employee/ officer</td></tr>
<tr><td>18</td><td>Number of people being liable to provide maintenance for</td><td>real</td><td>1, ..., 2</td></tr>
<tr><td>19</td><td>Telephone</td><td>enum</td><td>A191 : none
A192 : yes, registered under the customers name</td></tr>
<tr><td>20</td><td>foreign worker</td><td>enum</td><td>A201 : yes
A202 : no</td></tr>
</table>
</td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td>
<table>
<tr><th>class</th><th>samples</th><th>proportion (%)</th></tr>
<tr><td>good</td><td>700</td><td>70.00</td></tr>
<tr><td>bad</td><td>300</td><td>30.00</td></tr>
<tr><td>total</td><td>1000</td><td>100.00</td></tr>
</table>
</td></tr>
</table>

C.8 monks

The three monks problems were the basis of an international comparison of learning algorithms. The results of this comparison are summarised in (Thrun *et al.*, 1991). The artificial input domain, which is the same for all three problems, consists of six nominal attributes of robots. The concepts to be learned for each problem are:

monks1: head_shape = body_shape or jacket_colour = red

monks2: The second monk's problem is to determine whether exactly two of a robot's six attributes have their first value. There is no noise added.

monks3: (jacket_colour = green and holding = sword) or (jacket_colour != blue and body_shape != octagon).
5% class noise is added to the training set (6 misclassifications).

Rather than using three permutations of one problem, one permutation of each of the three problems is examined. Since there is no noise added to the test set for any of the problems, 100% accuracy is attainable in each case.

Table C.8: Details of the **monks** data sets.

<table>
<tr><td>dimensions</td><td>6</td></tr>
<tr><td>classes</td><td>2</td></tr>
<tr><td>samples</td><td>The test sample is the same for all three problems, consisting of all 432 possible examples. The training sample is different for each of the three data sets, being drawn randomly from the test set. The sample sizes are:
<table>
<tr><th>problem</th><th>training samples</th><th>N_{tr}</th><th>N_{val}</th></tr>
<tr><td><b>monks1</b></td><td>124</td><td>82</td><td>42</td></tr>
<tr><td><b>monks2</b></td><td>169</td><td>113</td><td>56</td></tr>
<tr><td><b>monks3</b></td><td>122</td><td>81</td><td>41</td></tr>
</table></td></tr>
<tr><td>preparation</td><td>Removed sample indices, moved class label from start to end of measurement vector.</td></tr>
<tr><td>partition</td><td>
<table>
<tr><th>Problem</th><th>N_{tr}</th><th>N_{val}</th><th>N_{tst}</th></tr>
<tr><td><b>monks1</b></td><td>82</td><td>42</td><td>432</td></tr>
<tr><td><b>monks2</b></td><td>113</td><td>56</td><td>432</td></tr>
<tr><td><b>monks3</b></td><td>81</td><td>41</td><td>432</td></tr>
</table></td></tr>
<tr><td>source</td><td>UCI</td></tr>
<tr><td>default error</td><td>50% for <b>monks1</b>, 32.87% for <b>monks2</b>, and 47.22% for <b>monks3</b>.</td></tr>
<tr><td>input attributes</td><td>
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>head shape</td><td>enum</td><td>round,square,octagon</td></tr>
<tr><td>2</td><td>body shape</td><td>enum</td><td>round,square,octagon</td></tr>
<tr><td>3</td><td>is smiling</td><td>bool</td><td>0,1</td></tr>
<tr><td>4</td><td>holding</td><td>enum</td><td>sword, balloon,flag</td></tr>
<tr><td>5</td><td>jacket colour</td><td>enum</td><td>red,yellow,green,blue</td></tr>
<tr><td>6</td><td>has tie</td><td>bool</td><td>0,1</td></tr>
</table></td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td>different for each class
<table>
<tr><th>Problem</th><th>Class 0</th><th>Class 1</th></tr>
<tr><td><b>monks1</b></td><td>216</td><td>216</td></tr>
<tr><td><b>monks2</b></td><td>290</td><td>142</td></tr>
<tr><td><b>monks3</b></td><td>204</td><td>228</td></tr>
</table></td></tr>
</table>

C.9 satimage

Satellite image classification problem. One frame of Landsat multi-spectral imagery consists of four digital images of the same scene in different spectral bands. Two of these are in the visible region (corresponding approximately to green and red regions of the visible spectrum) and two are in the (near) infra-red. Each pixel is an 8-bit binary word, with 0 corresponding to black and 255 to white. The spatial resolution of a pixel is about 80m × 80m. Each image contains 2340 × 3380 such pixels. The data is given in random order and certain lines of data have been removed so you cannot reconstruct the original image from this dataset.

Table C.9: Details of the **satimage** data set.

<table>
<tr><td>dimensions</td><td colspan="3">36</td></tr>
<tr><td>classes</td><td colspan="3">6</td></tr>
<tr><td>samples</td><td colspan="3">6435</td></tr>
<tr><td>preparation</td><td colspan="3">transformed class labels from $\{1, 2, 3, 4, 5, 7\}$ to $\{0, \ldots, 5\}$.</td></tr>
<tr><td>partition</td><td colspan="3">3217-1608-1610</td></tr>
<tr><td>source</td><td colspan="3">ELENA</td></tr>
<tr><td>default error</td><td colspan="3">76.18%</td></tr>
<tr><td>input attributes</td><td colspan="3">The present database is a (tiny) sub-area of a scene, consisting of 82 × 100 pixels. Each line of data corresponds to a 3×3 square neighbourhood of pixels completely contained within the 82×100 sub-area. Each line contains the pixel values in the four spectral bands (converted to ASCII) of each of the 9 pixels in the 3×3 neighbourhood and a number indicating the classification label of the central pixel. The aim is to predict this classification, given the multi-spectral values. The attributes are numerical, in the range 0 to 255 (8 bits).
In each line of data the four spectral values for the top-left pixel are given first followed by the four spectral values for the top-middle pixel and then those for the top-right pixel, and so on with the pixels read out in sequence left-to-right and top-to-bottom. Thus, the four spectral values for the central pixel are given by attributes 17, 18, 19 and 20.</td></tr>
<tr><td>missing values</td><td colspan="3">none</td></tr>
<tr><td rowspan="8">class distribution</td><td>class</td><td>samples</td><td>proportion(%)</td></tr>
<tr><td>red soil</td><td>1533</td><td>23.82</td></tr>
<tr><td>cotton crop</td><td>703</td><td>10.92</td></tr>
<tr><td>grey soil</td><td>1358</td><td>21.10</td></tr>
<tr><td>damp grey soil</td><td>626</td><td>9.73</td></tr>
<tr><td>soil with vegetation stubble</td><td>707</td><td>10.99</td></tr>
<tr><td>very damp grey soil</td><td>1508</td><td>23.43</td></tr>
<tr><td>total</td><td>6435</td><td>100.00</td></tr>
</table>

C.10 segment

Natural image segmentation problem. The instances were drawn randomly from a database of 7 outdoor images. The images were hand-segmented to create a classification for every pixel. Each instance is a 3 × 3 region.

Table C.10: Details of the **segment** data set.

<table>
<tr><td>dimensions</td><td>11 (originally 19)</td></tr>
<tr><td>classes</td><td>7</td></tr>
<tr><td>samples</td><td>2310</td></tr>
<tr><td>preparation</td><td>the constant and linearly-dependent attributes (3 and 10-16) were removed.</td></tr>
<tr><td>partition</td><td>1155-577-578</td></tr>
<tr><td>source</td><td>statlog</td></tr>
<tr><td>default error</td><td>85.71%</td></tr>
<tr><td>input attributes</td><td>the inputs are various statistical features of the 3 × 3 region. Eight attributes (3 and 10-16) are linear combinations of the data or are constant.
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>region-centroid-col</td><td>real</td><td>1...254</td></tr>
<tr><td>2</td><td>region-centroid-row</td><td>real</td><td>11...251</td></tr>
<tr><td>3</td><td>short-line-density-5</td><td>real</td><td>0...0.3</td></tr>
<tr><td>4</td><td>short-line-density-2</td><td>real</td><td>0...0.2</td></tr>
<tr><td>5</td><td>vedge-mean</td><td>real</td><td>0...29.2</td></tr>
<tr><td>6</td><td>vegde-sd</td><td>real</td><td>0...991.7</td></tr>
<tr><td>7</td><td>hedge-mean</td><td>real</td><td>0...44.7</td></tr>
<tr><td>8</td><td>hedge-sd</td><td>real</td><td>0...1386.3</td></tr>
<tr><td>9</td><td>value-mean</td><td>real</td><td>0...150.9</td></tr>
<tr><td>10</td><td>saturation-mean</td><td>real</td><td>0...1.0</td></tr>
<tr><td>11</td><td>hue-mean</td><td>real</td><td>−3.0442...2.9</td></tr>
</table></td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td>
<table>
<tr><th>class</th><th>samples</th><th>proportion (%)</th></tr>
<tr><td>brick face</td><td>330</td><td>14.29</td></tr>
<tr><td>sky</td><td>330</td><td>14.29</td></tr>
<tr><td>foliage</td><td>330</td><td>14.29</td></tr>
<tr><td>cement</td><td>330</td><td>14.29</td></tr>
<tr><td>window</td><td>330</td><td>14.29</td></tr>
<tr><td>path</td><td>330</td><td>14.29</td></tr>
<tr><td>grass</td><td>330</td><td>14.29</td></tr>
<tr><td>total</td><td>2310</td><td>100.00</td></tr>
</table></td></tr>
</table>

C.11 smoking

Data from a Survey on Attitudes Toward Smoking Legislation, analysed and described in (Bull, 1994). The task is to predict each subject's attitude toward restrictions on smoking in the work-place (prohibited, restricted or unrestricted) based on bylaw-related, smoking-related and sociodemographic attributes.

In a comparison of 33 classification methods on this data set (Lim *et al.*, 1997), no method was able to reduce the error rate below the default rate, indicating that there is no structure in the data to be learnt.

Table C.11: Details of the **smoking** data set.

<table>
<tr><td>dimensions</td><td>9</td></tr>
<tr><td>classes</td><td>3</td></tr>
<tr><td>samples</td><td>2855</td></tr>
<tr><td>preparation</td><td>removed example labels, moved class label to end of line, collapsed redundant encoding into one variable for work place and smoking status variables (removed 4 variables).</td></tr>
<tr><td>partition</td><td>1427-713-715</td></tr>
<tr><td>source</td><td>statlib</td></tr>
<tr><td>default error</td><td>30.47%</td></tr>
<tr><td>input attributes</td><td>
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>sampling weight</td><td>real</td><td>0.305 . . . 4.494</td></tr>
<tr><td>2</td><td>time of survey relative to by-law</td><td>enum</td><td>pre, post</td></tr>
<tr><td>3</td><td>place of work</td><td>enum</td><td>in_city, outside_city, not_outside_home</td></tr>
<tr><td>4</td><td>place of residence</td><td>enum</td><td>metro, city</td></tr>
<tr><td>5</td><td>smoking status</td><td>enum</td><td>never_smoked, quit_more_12_months, quit_6_to_12_months, quit_less_6_months, current_smoker</td></tr>
<tr><td>6</td><td>knowledge</td><td>real</td><td>0 . . . 12</td></tr>
<tr><td>7</td><td>sex</td><td>enum</td><td>female, male</td></tr>
<tr><td>8</td><td>(age in years - 50)/10</td><td>real</td><td>−3.2 . . . 4.5</td></tr>
<tr><td>9</td><td>education</td><td>enum</td><td>elementary, high_school, high_or_trade_school, college_or_uni, uni_degree</td></tr>
</table>
</td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td>
<table>
<tr><th>class</th><th>samples</th><th>proportion(%)</th></tr>
<tr><td>prohibited</td><td>151</td><td>5.29</td></tr>
<tr><td>restricted</td><td>719</td><td>25.18</td></tr>
<tr><td>unrestricted</td><td>1985</td><td>69.53</td></tr>
<tr><td>total</td><td>2855</td><td>100.00</td></tr>
</table>
</td></tr>
</table>

C.12 spirals

The task is to learn to discriminate between two sets of points which lie on two distinct spirals in the x-y plane. These spirals coil three times around the origin and around one another. This appears to be a very difficult task for many learning techniques. The training and validation data come from different regions of the spiral arms, while the test set contains points evenly distributed along the lengths of the spirals. The test set is plotted in Figure C.2.

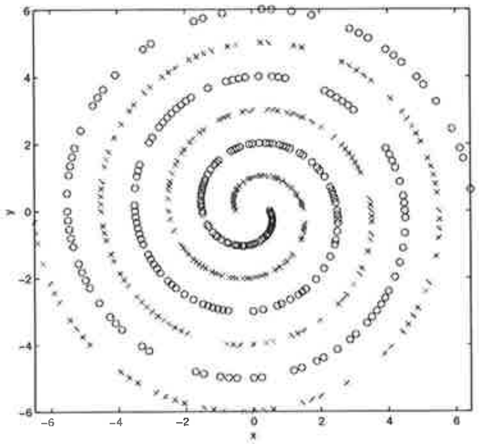


Figure C.2: Plot of the two interleaved **spirals** data set.

Table C.12: Details of the **spirals** data set.

<table>
<tr><td>dimensions</td><td>2</td></tr>
<tr><td>classes</td><td>2</td></tr>
<tr><td>samples</td><td>770</td></tr>
<tr><td>preparation</td><td>Used c program to generate data:
two-spirals_gen 4 6.5 > spirals.dat
Divided the training data into the training and validation sets. Removed commas, replaced "=> -" with 0 and "=> +" with 1.</td></tr>
<tr><td>partition</td><td>194-97-479</td></tr>
<tr><td>source</td><td>CMU</td></tr>
<tr><td>default error</td><td>48.23%</td></tr>
<tr><td>input attributes</td><td><table><tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr><tr><td>1</td><td>x</td><td>real</td><td>−6, . . . , 6</td></tr><tr><td>2</td><td>y</td><td>real</td><td>−6, . . . , 6</td></tr></table></td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td><table><tr><th>class</th><th>samples</th><th>proportion(%)</th></tr><tr><td>0</td><td>248</td><td>51.77</td></tr><tr><td>1</td><td>231</td><td>48.23</td></tr><tr><td>total</td><td>479</td><td>100.00</td></tr></table></td></tr>
</table>

C.13 titanic

The titanic dataset gives the values of three categorical attributes for each of the 2201 people on board the Titanic when it struck an iceberg and sank. The attributes are social class (first class, second class, third class, crew member), age (adult or child), and sex. The task is to predict whether or not the person survived.

The question of interest for this natural data set is how survival relates to the other attributes. There is obviously no practical need to predict survival, so the real interest is in interpretation, but success at prediction would appear to be closely related to the discovery of interesting features of the relationship. Note that there are only sixteen possible combinations of input attributes for this prediction task, so the interesting behaviour will be that with small training sets.

Table C.13: Details of the **titanic** data set.

<table>
<tr><td>dimensions</td><td>3</td></tr>
<tr><td>classes</td><td>2</td></tr>
<tr><td>samples</td><td>2201</td></tr>
<tr><td>preparation</td><td>replaced class-type 1st, 2nd, 3rd, crew with 0, 1, 2, 3. Replaced adult, child with 0, 1. Replaced male, female with 0, 1. Replaced survived? no, yes with 0, 1.</td></tr>
<tr><td>partition</td><td>1100-550-551</td></tr>
<tr><td>source</td><td>delve</td></tr>
<tr><td>default error</td><td>32.30%</td></tr>
<tr><td>input attributes</td><td><table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>class</td><td>enum</td><td>1st, 2nd, 3rd, crew</td></tr>
<tr><td>2</td><td>age</td><td>enum</td><td>child, adult</td></tr>
<tr><td>3</td><td>sex</td><td>enum</td><td>female, male</td></tr>
</table></td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td><table>
<tr><th>class</th><th>samples</th><th>proportion(%)</th></tr>
<tr><td>died</td><td>1490</td><td>67.70</td></tr>
<tr><td>survived</td><td>711</td><td>32.30</td></tr>
<tr><td>total</td><td>2201</td><td>100.00</td></tr>
</table></td></tr>
</table>

C.14 vehicle

Vehicle silhouette dataset. The purpose is to classify a given silhouette as one of four types of vehicle, using a set of features extracted from the silhouette. The vehicle may be viewed from one of many different angles.

The features were extracted from the silhouettes by the HIPS (Hierarchical Image Processing System) extension BINATTS, which extracts a combination of scale independent features utilising both classical moments based measures such as scaled variance, skewness and kurtosis about the major/minor axes and heuristic measures such as hollows, circularity, rectangularity and compactness. Four "Corgie" model vehicles were used for the experiment: a double decker bus, Cheverolet van, Saab 9000 and an Opel Manta 400. This particular combination of vehicles was chosen with the expectation that the bus, van and either one of the cars would be readily distinguishable, but it would be more difficult to distinguish between the cars.

Table C.14: Details of the **vehicle** data set.

<table>
<tr><td>dimensions</td><td>18</td></tr>
<tr><td>classes</td><td>4</td></tr>
<tr><td>samples</td><td>846</td></tr>
<tr><td>preparation</td><td>subtracted 1 from class labels.</td></tr>
<tr><td>partition</td><td>423-211-212</td></tr>
<tr><td>source</td><td>statlog</td></tr>
<tr><td>default error</td><td>76.48%</td></tr>
<tr><td>input attributes</td><td>
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>compactness</td><td>real</td><td>$73,\ldots,119$</td></tr>
<tr><td>2</td><td>circularity</td><td>real</td><td>$33,\ldots,59$</td></tr>
<tr><td>3</td><td>distance circularity</td><td>real</td><td>$40,\ldots,112$</td></tr>
<tr><td>4</td><td>radius ratio</td><td>real</td><td>$104,\ldots,333$</td></tr>
<tr><td>5</td><td>pr.axis aspect ratio</td><td>real</td><td>$47,\ldots,138$</td></tr>
<tr><td>6</td><td>max.length aspect ratio</td><td>real</td><td>$2,\ldots,55$</td></tr>
<tr><td>7</td><td>scatter ratio</td><td>real</td><td>$112,\ldots,265$</td></tr>
<tr><td>8</td><td>elongatedness</td><td>real</td><td>$26,\ldots,61$</td></tr>
<tr><td>9</td><td>pr.axis rectangularity</td><td>real</td><td>$17,\ldots,29$</td></tr>
<tr><td>10</td><td>max.length rectangularity</td><td>real</td><td>$118,\ldots,188$</td></tr>
<tr><td>11</td><td>scaled variance</td><td>real</td><td>$130,\ldots,320$</td></tr>
<tr><td>12</td><td>scaled variance</td><td>real</td><td>$184,\ldots,1018$</td></tr>
<tr><td>13</td><td>scaled radius of gyration</td><td>real</td><td>$109,\ldots,268$</td></tr>
<tr><td>14</td><td>skewness about</td><td>real</td><td>$59,\ldots,135$</td></tr>
<tr><td>15</td><td>skewness about</td><td>real</td><td>$0,\ldots,22$</td></tr>
<tr><td>16</td><td>kurtosis about</td><td>real</td><td>$0,\ldots,41$</td></tr>
<tr><td>17</td><td>kurtosis about</td><td>real</td><td>$176,\ldots,206$</td></tr>
<tr><td>18</td><td>hollows ratio</td><td>real</td><td>$181,\ldots,211$</td></tr>
</table>
</td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td>
<table>
<tr><th>class</th><th>samples</th><th>proportion(%)</th></tr>
<tr><td>opel</td><td>212</td><td>25.06</td></tr>
<tr><td>saab</td><td>217</td><td>25.65</td></tr>
<tr><td>bus</td><td>218</td><td>25.77</td></tr>
<tr><td>van</td><td>199</td><td>23.52</td></tr>
<tr><td>total</td><td>846</td><td>100.00</td></tr>
</table>
</td></tr>
</table>

C.15 yeast

Predict the localisation site of a protein.

Table C.15: Details of the **yeast** data set.

<table>
<tr><td>dimensions</td><td>8</td></tr>
<tr><td>classes</td><td>10</td></tr>
<tr><td>samples</td><td>1484</td></tr>
<tr><td>preparation</td><td>removed first attribute (label). Replaced mnemonic class labels with numeric starting at 0:
<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td>CYT</td><td>NUC</td><td>MIT</td><td>ME3</td><td>ME2</td><td>ME1</td><td>EXC</td><td>VAC</td><td>POX</td><td>ERL</td></tr>
</table></td></tr>
<tr><td>partition</td><td>742-371-371</td></tr>
<tr><td>source</td><td>UCI</td></tr>
<tr><td>default error</td><td>68.80%</td></tr>
<tr><td>input attributes</td><td>
<table>
<tr><th>attr.</th><th>name</th><th>type</th><th>range</th></tr>
<tr><td>1</td><td>mcg: McGeoch's method for signal sequence recognition.</td><td>real</td><td>0.1100, . . . , 1.0000</td></tr>
<tr><td>2</td><td>gvh: von Heijne's method for signal sequence recognition.</td><td>real</td><td>0.1300, . . . , 1.0000</td></tr>
<tr><td>3</td><td>alm: Score of the ALOM membrane spanning region prediction program.</td><td>real</td><td>0.2100, . . . , 1.0000</td></tr>
<tr><td>4</td><td>mit: Score of discriminant analysis of the amino acid content of the N-terminal region (20 residues long) of mitochondrial and non-mitochondrial proteins.</td><td>real</td><td>0, . . . , 1.0000</td></tr>
<tr><td>5</td><td>erl: Presence of "HDEL" substring (thought to act as a signal for retention in the endoplasmic reticulum lumen). Binary attribute.</td><td>real</td><td>0.5000, . . . , 1.0000</td></tr>
<tr><td>6</td><td>pox: Peroxisomal targeting signal in the C-terminus.</td><td>real</td><td>0, . . . , 0.8300</td></tr>
<tr><td>7</td><td>vac: Score of discriminant analysis of the amino acid content of vacuolar and extracellular proteins.</td><td>real</td><td>0, . . . , 0.7300</td></tr>
<tr><td>8</td><td>nuc: Score of discriminant analysis of nuclear localization signals of nuclear and non-nuclear proteins.</td><td>real</td><td>0, . . . , 1.0000</td></tr>
</table></td></tr>
<tr><td>missing values</td><td>none</td></tr>
<tr><td>class distribution</td><td>
<table>
<tr><th>class</th><th>samples</th><th>proportion(%)</th></tr>
<tr><td>CYT (cytosolic or cytoskeletal)</td><td>463</td><td>31.1995</td></tr>
<tr><td>NUC (nuclear)</td><td>429</td><td>28.9084</td></tr>
<tr><td>MIT (mitochondrial)</td><td>244</td><td>16.4420</td></tr>
<tr><td>ME3 (membrane protein, no N-terminal signal)</td><td>163</td><td>10.9838</td></tr>
<tr><td>ME2 (membrane protein, uncleaved signal)</td><td>51</td><td>3.4367</td></tr>
<tr><td>ME1 (membrane protein, cleaved signal)</td><td>44</td><td>2.9650</td></tr>
<tr><td>EXC (extracellular)</td><td>37</td><td>2.4933</td></tr>
<tr><td>VAC (vacuolar)</td><td>30</td><td>2.0216</td></tr>
<tr><td>POX (peroxisomal)</td><td>20</td><td>1.3477</td></tr>
<tr><td>ERL (endoplasmic reticulum lumen)</td><td>5</td><td>0.3369</td></tr>
<tr><td>total</td><td>1484</td><td>100.00</td></tr>
</table></td></tr>
</table>

Appendix D

EPrep Parameters

This Appendix contains the parameter files used by EPrep for the experiments of Chapter 6. The parameter files are listed here for repeatability of the experiments. If these parameters are used with the same random seed chosen by EPrep at the time of execution, identical results will be obtained. The random seed used for each problem can be found in the HTML reports included on the CD-rom; see Appendix A.

The parameters were essentially the same for all permutations of a given data set. Therefore only the parameter file for the first permutation is included here. As an exception, different parameters were used for each of the three **monks** problems, and so they are listed separately. For guidelines on the selection of parameters for EPrep, refer to Section 5.11.

D.1 abalone

```
// EPrep Parameter File        Mon May 11 21:43:06 1998

data = /home/users/jsherrah/EPrep3/data/abalone_1.dat  (tr=2088 va=1044)
classifier = MDTM
classifier = PPD
classifier = ML
tournament_size = 2
rat_delta = 0.02
rat_n_init = 750
rat_n_inc = 40
n_sort = 10
n_runs = 10
pop_size = 400
n_gens = 30
rand_seed = 0
n_reproduction = 70
seed_init_pop = True
optimise_pop = True
opt_max_iter = 7
opt_convergence_val = 0.01
gl_thresh = 15
tp_thresh = 0.001
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 9
max_run_depth = 9
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
function = Add
function = Subtract
function = Multiply
function = Divide
function = Equal To
function = Less Than
function = If-Then-Else
function = And
function = Or
function = Not
```

D.2 australian credit

```
// EPrep Parameter File        Tue May 12 12:44:56 1998
data = /home/users/jsherrah/EPrep3/data/australian_1.dat  (tr=345 va=172)
classifier = MDTM
classifier = PPD
tournament_size = 2
rat_delta = 0.02
rat_n_init = 120
rat_n_inc = 8
n_sort = 22
n_runs = 10
pop_size = 500
n_gens = 50
rand_seed = 0
n_reproduction = 26
seed_init_pop = True
optimise_pop = True
opt_max_iter = 15
opt_convergence_val = 0.01
gl_thresh = 20
tp_thresh = 0.01
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 8
max_run_depth = 8
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
terminal = Zero
terminal = One
terminal = Two
terminal = Three
function = Add
function = Subtract
function = Multiply
function = Divide
function = Abs
function = Log
function = Equal To
function = Less Than
function = If-Then-Else
function = And
function = Or
function = Not
```

D.3 balance

```
// EPrep Parameter File        Tue May 12 11:11:38 1998

data = /home/cssip/jsherrah/EPrep3/data/balance_1.dat  (tr=312 va=156)
classifier = MDTM
classifier = PPD
tournament_size = 2
rat_delta = 0.02
rat_n_init = 90
rat_n_inc = 6
n_sort = 18
n_runs = 10
pop_size = 400
n_gens = 40
rand_seed = 0
n_reproduction = 7
seed_init_pop = True
optimise_pop = True
opt_max_iter = 15
opt_convergence_val = 0.001
gl_thresh = 35
tp_thresh = 0.001
tr_strip_len = 8
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 8
max_run_depth = 8
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
function = Add
function = Subtract
function = Multiply
function = Divide
function = Abs
function = Equal To
function = Less Than
function = If-Then-Else
function = And
function = Or
function = Not
```

D.4 concentric

```
// EPrep Parameter File        Mon May 11 15:21:39 1998

data = /home/cssip_a/jsherrah/data/concentric_1.dat  (tr=1250 va=625)
classifier = MDTM
classifier = PPD
tournament_size = 18
rat_delta = 0.02
rat_n_init = 800
rat_n_inc = 20
n_sort = 10
n_runs = 10
pop_size = 600
n_gens = 50
rand_seed = 0
n_reproduction = 100
seed_init_pop = True
optimise_pop = True
opt_max_iter = 10
opt_convergence_val = 0.005
gl_thresh = 100
tp_thresh = 0.01
tr_strip_len = 20
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 15
max_run_depth = 17
use_h_l_introns = True
prob_inversion = 0
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
terminal = One
terminal = Two
function = Add
function = Subtract
function = Multiply
function = Divide
```

D.5 contraceptive method choice (cmc)

```
// EPrep Parameter File        Tue May 12 11:19:53 1998

data = /home/cssip/jsherrah/EPrep3/data/cmc_1.dat  (tr=736 va=368)
classifier = MDTM
tournament_size = 3
rat_delta = 0.02
rat_n_init = 240
rat_n_inc = 12
n_sort = 33
n_runs = 10
pop_size = 500
n_gens = 50
rand_seed = 0
n_reproduction = 150
seed_init_pop = True
optimise_pop = True
opt_max_iter = 10
opt_convergence_val = 0.01
gl_thresh = 50
tp_thresh = 0.015
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 8
max_run_depth = 8
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
terminal = Zero
terminal = One
terminal = Two
terminal = Three
function = Add
function = Subtract
function = Multiply
function = Divide
function = Abs
function = Equal To
function = Less Than
function = If-Then-Else
function = And
function = Or
function = Not
```

D.6 diabetes

```
// EPrep Parameter File        Mon May 11 14:36:06 1998

data = /home/cssip/jsherrah/EPrep3/data/diabetes_1.dat  (tr=384 va=192)
classifier = MDTM
classifier = PPD
tournament_size = 2
rat_delta = 0.02
rat_n_init = 100
rat_n_inc = 6
n_sort = 28
n_runs = 10
pop_size = 500
n_gens = 40
rand_seed = 0
n_reproduction = 25
seed_init_pop = True
optimise_pop = True
opt_max_iter = 15
opt_convergence_val = 0.01
gl_thresh = 50
tp_thresh = 0.015
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 5
max_init_depth = 7
max_run_depth = 7
use_h_l_introns = True
prob_inversion = 1.0
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
function = Add
function = Subtract
function = Multiply
function = Divide
function = Abs
function = Log
function = Less Than
function = If-Then-Else
```

D.7 german credit

```
// EPrep Parameter File        Tue May 12 13:45:49 1998

data = /home/cssip/jsherrah/EPrep3/data/german_1.dat  (tr=500 va=250)
classifier = MDTM
classifier = ML
tournament_size = 2
rat_delta = 0.02
rat_n_init = 250
rat_n_inc = 6
n_sort = 32
n_runs = 10
pop_size = 500
n_gens = 50
rand_seed = 0
n_reproduction = 50
seed_init_pop = True
optimise_pop = True
opt_max_iter = 10
opt_convergence_val = 0.01
gl_thresh = 30
tp_thresh = 0.01
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 5
max_init_depth = 8
max_run_depth = 13
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
function = Add
function = Subtract
function = Multiply
function = Divide
function = Equal To
function = Less Than
function = If-Then-Else
function = And
function = Or
function = Not
```

D.8 monks1

```
// EPrep Parameter File        Tue May 12 16:00:27 1998

data = /home/cssip/jsherrah/EPrep3/data/monks_1.dat  (tr=82 va=42)
classifier = ML
classifier = MDTM
classifier = PPD
tournament_size = 2
rat_delta = 0.01
rat_n_init = 82
rat_n_inc = 0
n_sort = 0
n_runs = 10
pop_size = 400
n_gens = 30
rand_seed = 0
n_reproduction = 28
seed_init_pop = True
optimise_pop = True
opt_max_iter = 15
opt_convergence_val = 0.01
gl_thresh = 60
tp_thresh = 0.01
tr_strip_len = 8
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 7
max_run_depth = 7
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
function = Equal To
function = If-Then-Else
function = And
function = Or
function = Not
```

D.9 monks2

```
// EPrep Parameter File        Mon Mar 16 13:22:08 1998

data = /home/cssip/jsherrah/EPrep3/data/monks_2.dat  (tr=113 va=56)
classifier = MDTM
classifier = PPD
classifier = ML
tournament_size = 7
rat_delta = 0.01
rat_n_init = 113
rat_n_inc = 1
n_sort = 0
n_runs = 10
pop_size = 500
n_gens = 30
rand_seed = 0
n_reproduction = 10
seed_init_pop = True
optimise_pop = True
opt_max_iter = 10
opt_convergence_val = 0.4
gl_thresh = 10
tp_thresh = 0.01
tr_strip_len = 8
feature_corr = True
results_to_log = All
results_log_freq = 5
max_init_depth = 7
max_run_depth = 7
use_h_l_introns = True
prob_inversion = 0.5
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
function = Equal To
function = If-Then-Else
function = And
function = Or
function = Not
```

D.10 monks3

```
// EPrep Parameter File        Tue May 12 16:12:41 1998

data = /home/cssip/jsherrah/EPrep3/data/monks_3.dat  (tr=81 va=41)
classifier = MDTM
classifier = PPD
tournament_size = 2
rat_delta = 0.01
rat_n_init = 81
rat_n_inc = 0
n_sort = 0
n_runs = 10
pop_size = 400
n_gens = 30
rand_seed = 0
n_reproduction = 20
seed_init_pop = True
optimise_pop = False
opt_max_iter = 15
opt_convergence_val = 0.01
gl_thresh = 70
tp_thresh = 0.01
tr_strip_len = 8
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 7
max_run_depth = 7
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
function = Equal To
function = If-Then-Else
function = And
function = Or
function = Not
```

D.11 satimage

```
// EPrep Parameter File        Tue May 12 12:07:39 1998
data = /home/cssip/jsherrah/EPrep3/data/satimage_1.dat  (tr=3217 va=1608)
classifier = MDTM
classifier = PPD
tournament_size = 2
rat_delta = 0.02
rat_n_init = 600
rat_n_inc = 60
n_sort = 84
n_runs = 10
pop_size = 500
n_gens = 50
rand_seed = 0
n_reproduction = 116
seed_init_pop = True
optimise_pop = True
opt_max_iter = 10
opt_convergence_val = 0.001
gl_thresh = 100
tp_thresh = 0.001
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 10
max_run_depth = 10
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
terminal = Zero
terminal = One
terminal = Two
terminal = Three
function = Add
function = Subtract
function = Multiply
function = Divide
function = Abs
function = Log
function = Equal To
function = Less Than
function = If-Then-Else
function = And
function = Or
function = Not
```

D.12 segment

```
// EPrep Parameter File        Tue May 12 11:33:03 1998

data = /home/cssip/jsherrah/EPrep3/data/segment_1.dat  (tr=1155 va=577)
classifier = ML
tournament_size = 2
rat_delta = 0.02
rat_n_init = 420
rat_n_inc = 21
n_sort = 21
n_runs = 10
pop_size = 500
n_gens = 40
rand_seed = 0
n_reproduction = 8
seed_init_pop = True
optimise_pop = True
opt_max_iter = 15
opt_convergence_val = 0.001
gl_thresh = 100
tp_thresh = 0.001
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 8
max_run_depth = 8
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
terminal = Zero
terminal = One
terminal = Two
terminal = Three
function = Add
function = Subtract
function = Multiply
function = Divide
function = Abs
function = Equal To
function = Less Than
function = If-Then-Else
function = And
function = Or
function = Not
```

D.13 smoking

```
// EPrep Parameter File        Tue May 12 15:39:48 1998

data = /home/cssip/jsherrah/EPrep3/data/smoking_1.dat  (tr=1427 va=713)
classifier = MDTM
classifier = PPD
classifier = ML
tournament_size = 2
rat_delta = 0.02
rat_n_init = 450
rat_n_inc = 30
n_sort = 66
n_runs = 10
pop_size = 500
n_gens = 30
rand_seed = 0
n_reproduction = 35
seed_init_pop = True
optimise_pop = True
opt_max_iter = 15
opt_convergence_val = 0.01
gl_thresh = 100
tp_thresh = 0.001
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 5
max_init_depth = 8
max_run_depth = 8
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
function = Subtract
function = Multiply
function = Divide
function = Equal To
function = If-Then-Else
function = And
function = Or
function = Not
```

D.14 spirals

```
// EPrep Parameter File        Mon May 11 21:19:38 1998

data = /home/cssip/jsherrah/EPrep3/data/spirals_1.dat  (tr=194 va=97)
classifier = MDTM
classifier = PPD
classifier = ML
tournament_size = 18
rat_delta = 0.02
rat_n_init = 194
rat_n_inc = 0
n_sort = 0
n_runs = 10
pop_size = 800
n_gens = 100
rand_seed = 0
n_reproduction = 100
seed_init_pop = True
optimise_pop = True
opt_max_iter = 10
opt_convergence_val = 0.001
gl_thresh = 100
tp_thresh = 0.01
tr_strip_len = 25
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 15
max_run_depth = 17
use_h_l_introns = True
prob_inversion = 0
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
terminal = Pi
function = Add
function = Subtract
function = Multiply
function = Divide
function = Sin
function = Cos
```

D.15 titanic

```
// EPrep Parameter File        Mon May 11 15:00:37 1998

data = /home/cssip_a/jsherrah/data/titanic_1.dat  (tr=1100 va=550)
classifier = MDTM
classifier = PPD
tournament_size = 2
rat_delta = 0.01
rat_n_init = 400
rat_n_inc = 20
n_sort = 80
n_runs = 10
pop_size = 500
n_gens = 40
rand_seed = 0
n_reproduction = 250
seed_init_pop = True
optimise_pop = True
opt_max_iter = 12
opt_convergence_val = 0.001
gl_thresh = 25
tp_thresh = 0.001
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 1
max_init_depth = 7
max_run_depth = 7
use_h_l_introns = True
prob_inversion = 1.0
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
function = Equal To
function = If-Then-Else
function = And
function = Or
function = Not
```

D.16 vehicle

```
// EPrep Parameter File        Tue May 12 14:24:16 1998

data = /home/cssip/jsherrah/EPrep3/data/vehicle_1.dat  (tr=423 va=211)
classifier = MDTM
classifier = ML
classifier = PPD
tournament_size = 2
rat_delta = 0.02
rat_n_init = 200
rat_n_inc = 4
n_sort = 12
n_runs = 10
pop_size = 500
n_gens = 50
rand_seed = 0
n_reproduction = 16
seed_init_pop = True
optimise_pop = True
opt_max_iter = 15
opt_convergence_val = 0.01
gl_thresh = 60
tp_thresh = 0.01
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 5
max_init_depth = 9
max_run_depth = 9
use_h_l_introns = True
prob_inversion = 1
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
function = Add
function = Subtract
function = Multiply
function = Divide
function = Abs
function = Log
function = Exp
function = Less Than
function = If-Then-Else
```

D.17 yeast

```
// EPrep Parameter File

data = /home/cssip/jsherrah/EPrep3/data/yeast_1.dat  (tr=742 va=371)
classifier = PPD
classifier = ML
tournament_size = 2
rat_delta = 0.02
rat_n_init = 350
rat_n_inc = 10
n_sort = 10
n_runs = 10
pop_size = 600
n_gens = 150
rand_seed = 0
n_reproduction = 100
seed_init_pop = True
optimise_pop = False
opt_max_iter = 10
opt_convergence_val = 0.01
gl_thresh = 50
tp_thresh = 0.015
tr_strip_len = 10
feature_corr = True
results_to_log = All
results_log_freq = 5
max_init_depth = 7
max_run_depth = 7
use_h_l_introns = True
prob_inversion = 1.0
operator = High-Level Crossover
operator = Grow Mutation
operator = One-Node Mutation
operator = All-Nodes Mutation
operator = One-Symbol Mutation
operator = Swap Mutation
operator = Truncate Mutation
operator = Hoist Mutation
operator = Add-Feature Mutation
operator = Delete-Feature Mutation
terminal = Random Ephemeral Constant
function = Add
function = Subtract
function = Multiply
function = Divide
function = Abs
function = Log
```

Appendix E

Results of EPrep Experiments

This Appendix contains the plots obtained by running EPrep on each of the 15 public-domain data sets used in the experiments of Chapter 6. Each data set has a sub-section containing plots of the following 10 quantities versus generation:

best-of-generation fitness: the minimum objective function value (RAT training fitness) of individuals in the population at generation g. Fitness is based on training set classification error, so lower is better.

best-of-generation validation error: the validation set error of the best-of-generation individual at generation g.

average fitness: the mean objective function value of individuals in the population at generation g.

standard deviation of fitness: the standard deviation of objective function values of individuals in the population at generation g.

average number of features: the average number of features contained in individuals in the population at generation g.

average number of nodes: the average number of nodes contained in individuals in the population at generation g.

average number of introns: the average number of introns contained in individuals in the population at generation g.

average number of RAT trials: the average number of training samples required to evaluate the fitness of individuals in the population at generation g.

average optimisation improvement: the average improvement in fitness brought about by local optimisation at generation g.

average operator probabilities: the average probability per genetic operator contained in individuals in the population at generation g.

All results are averaged over the 10 runs performed for each permutation. Each plot shows the results for all three permutations of the data on the same axes. The only exception is the average operator probabilities: only the results for the first permutation of the data are shown, because the plot would become too complicated if the other two permutations were added.

Note that runs generally last for a different number of generations on the same permutation of the data, so when a value was absent from a shorter run, the value at the last generation for that run was used to calculate the average. For example, if the longest run lasted 20 generations, and run i lasted 15 generations, then we set $q(j) = q(i), j = 16, \ldots 20$ to calculate the average, where $q(g)$ is the value of the quantity at generation g. Since these plots are averaged, fluctuations in the individual run-curves cannot be observed. Nevertheless the averaged plots exhibit some interesting fluctuations.

E.1 abalone

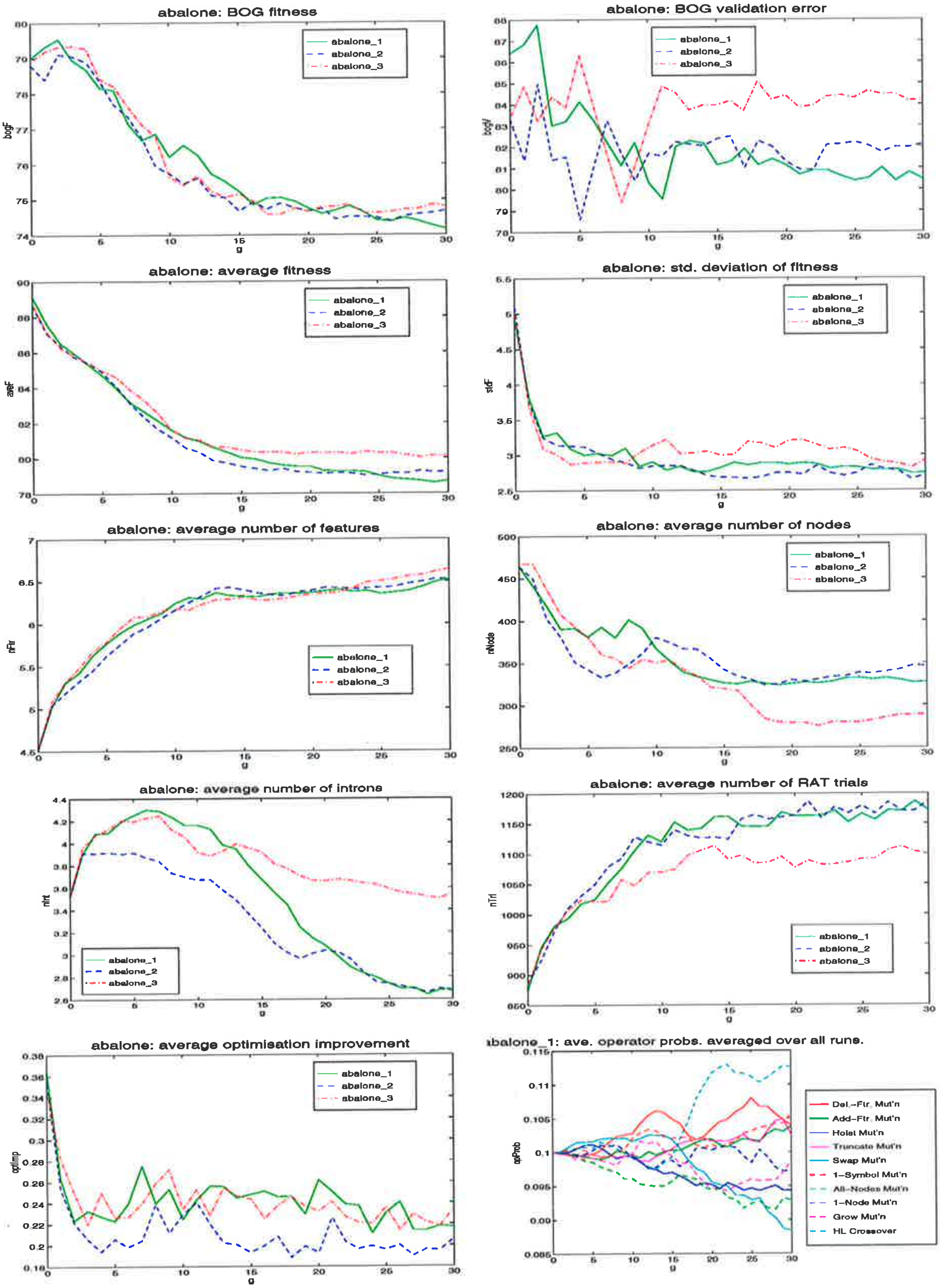


Figure E.1: Performance measures for **abalone**.

E.2 australian credit

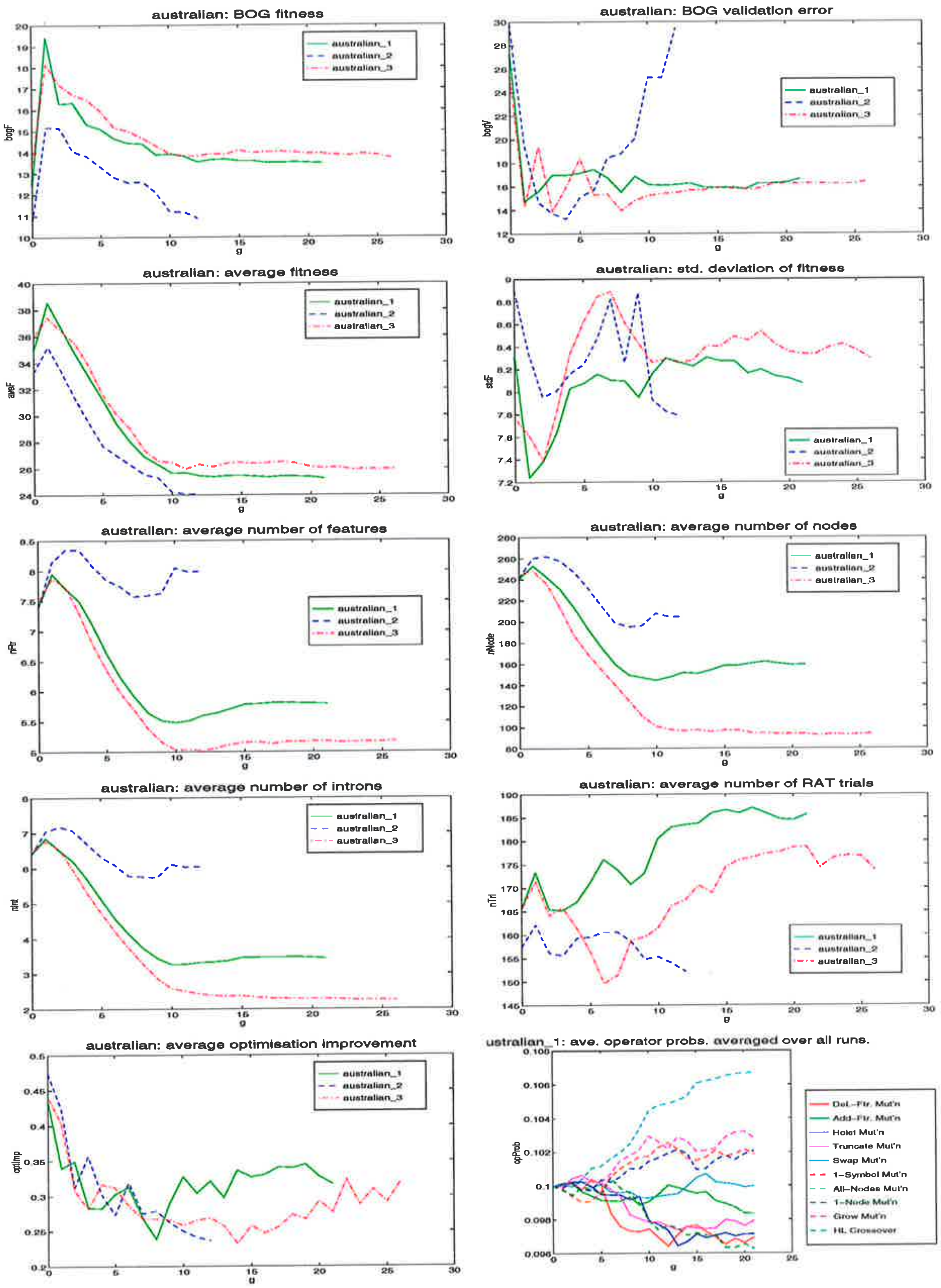


Figure E.2: Performance measures for **australian**.

E.3 balance

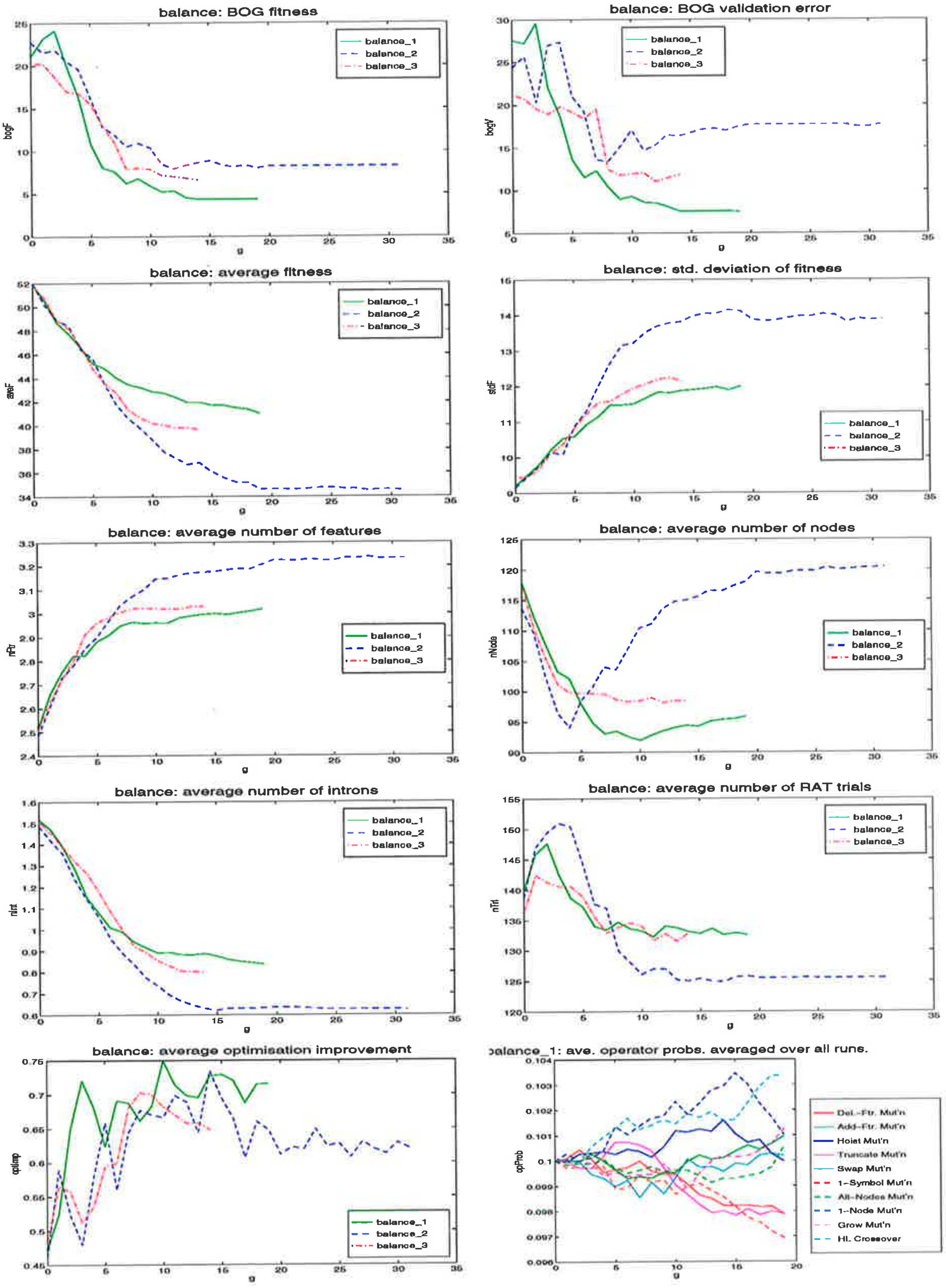


Figure E.3: Performance measures for **balance**.

E.4 concentric

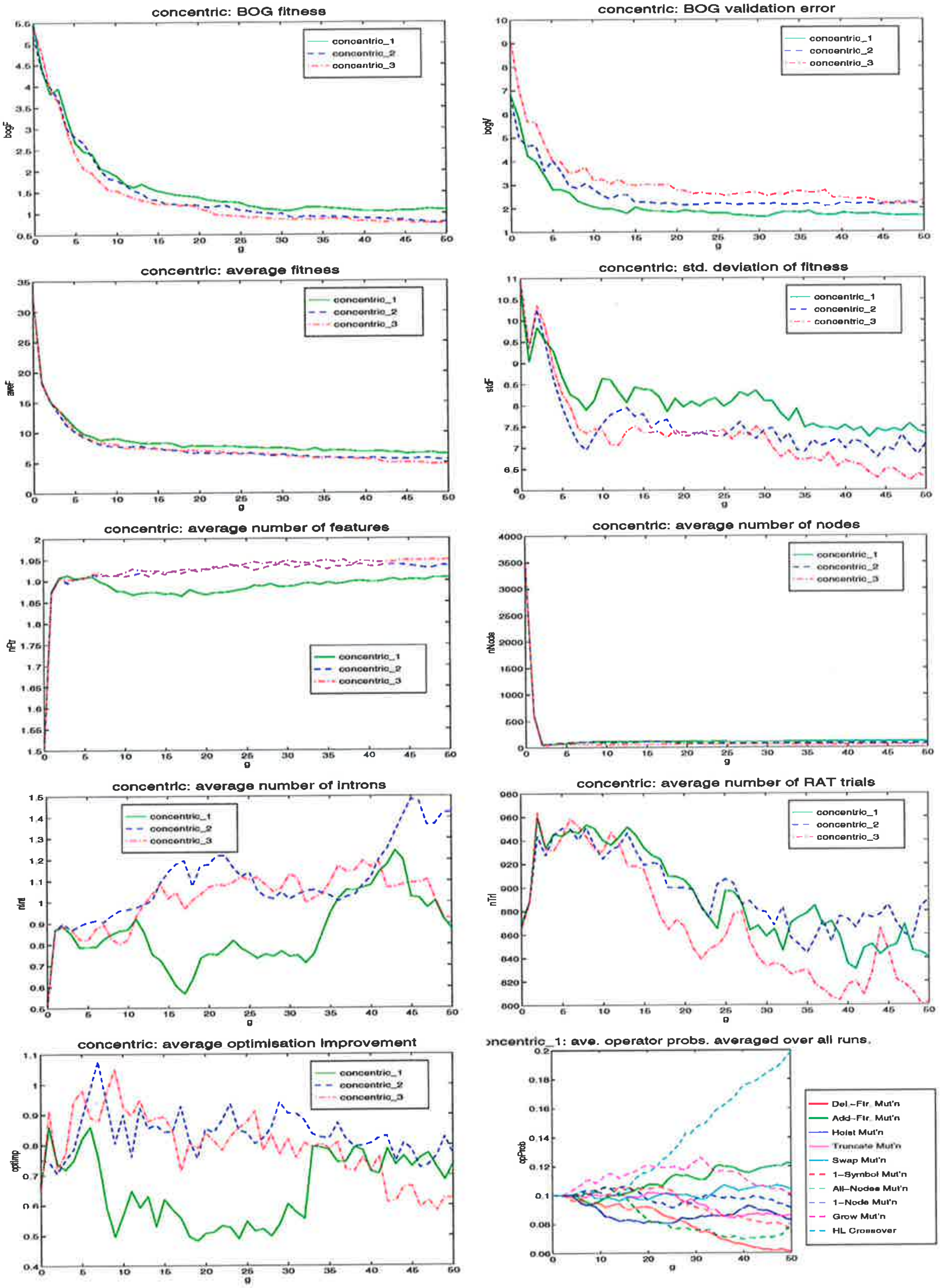


Figure E.4: Performance measures for **concentric**.

E.5 contraceptive method choice (cmc)

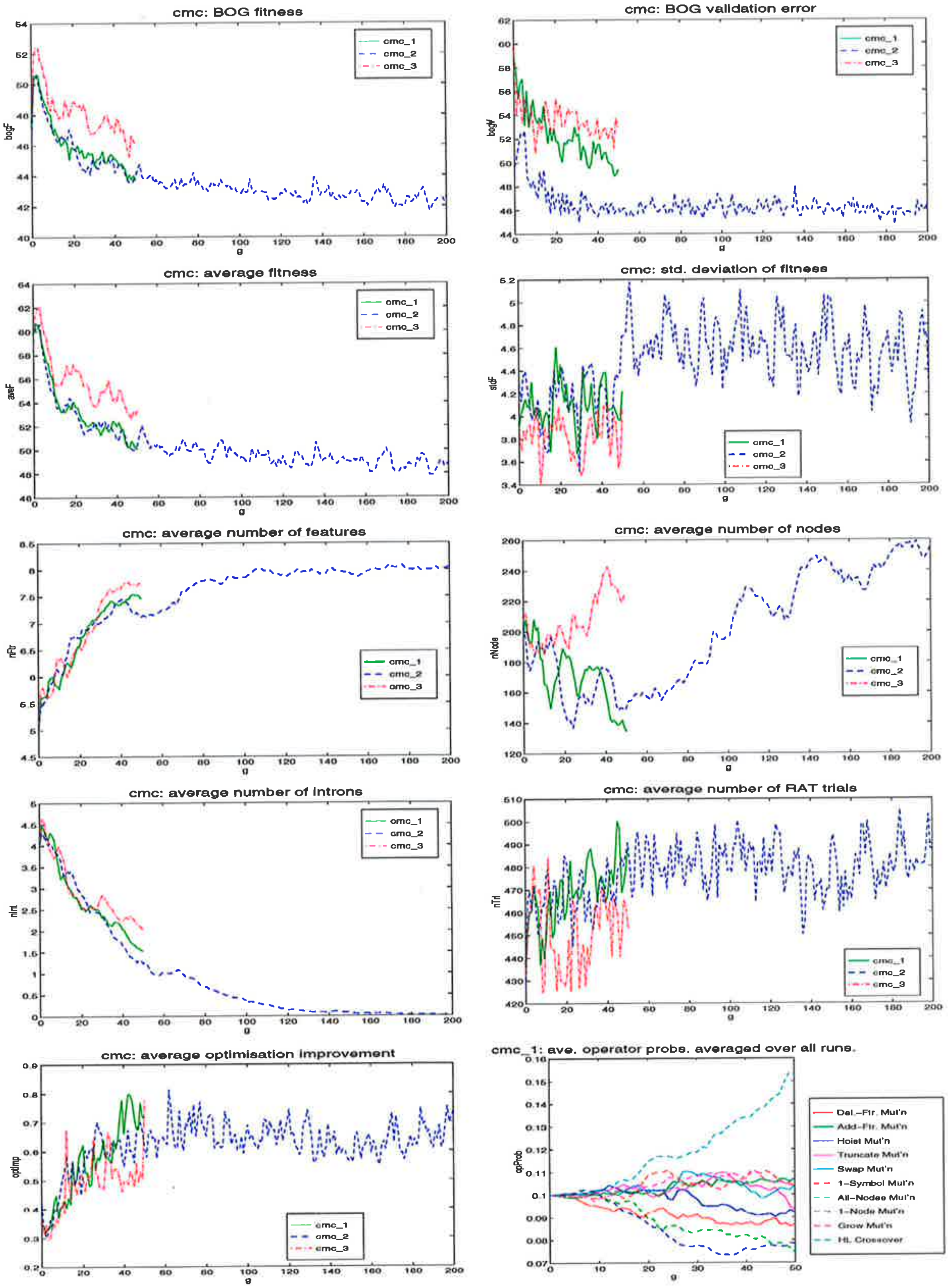


Figure E.5: Performance measures for **cmc**.

E.6 diabetes

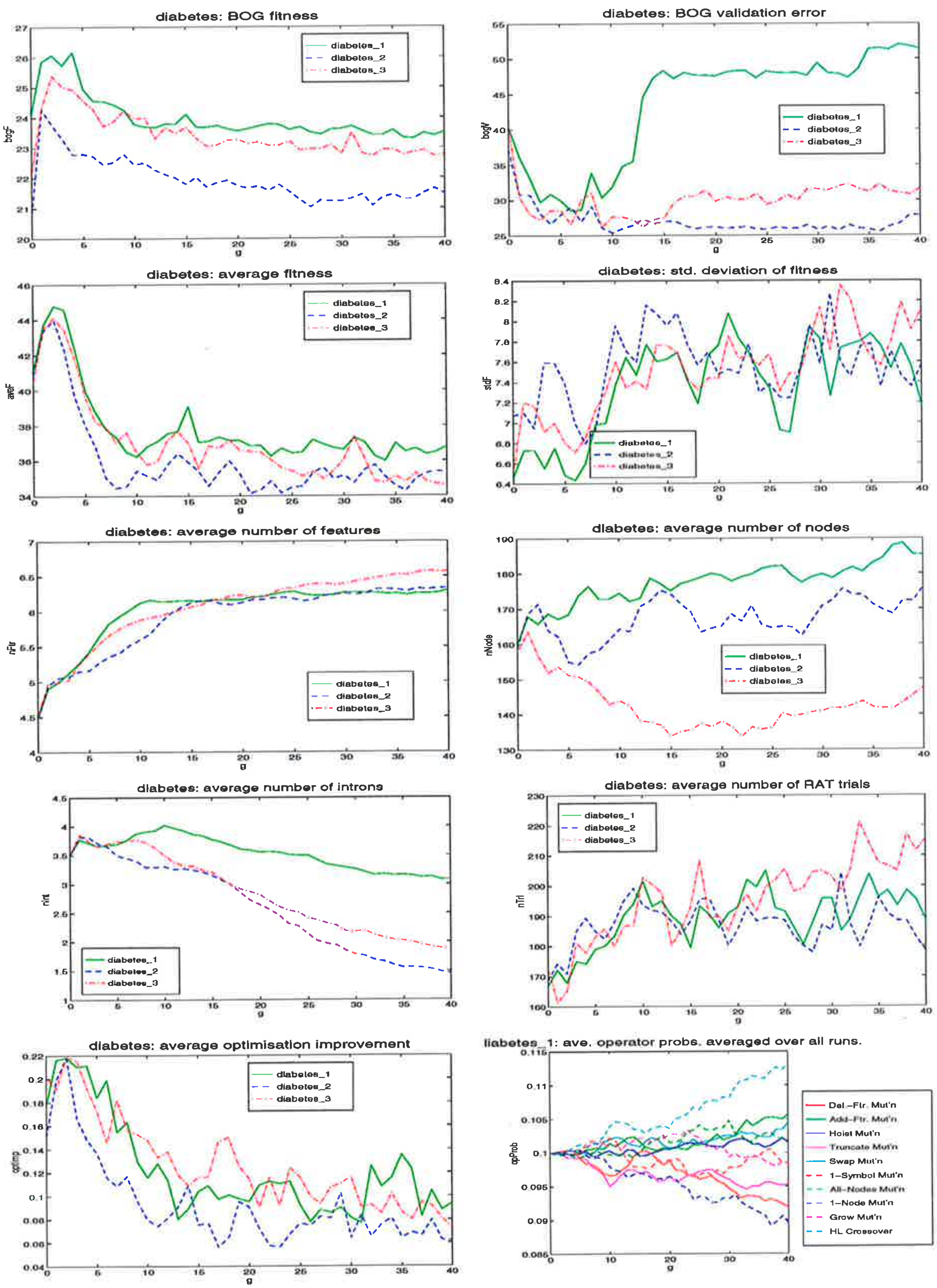


Figure E.6: Performance measures for **diabetes**.

E.7 german credit

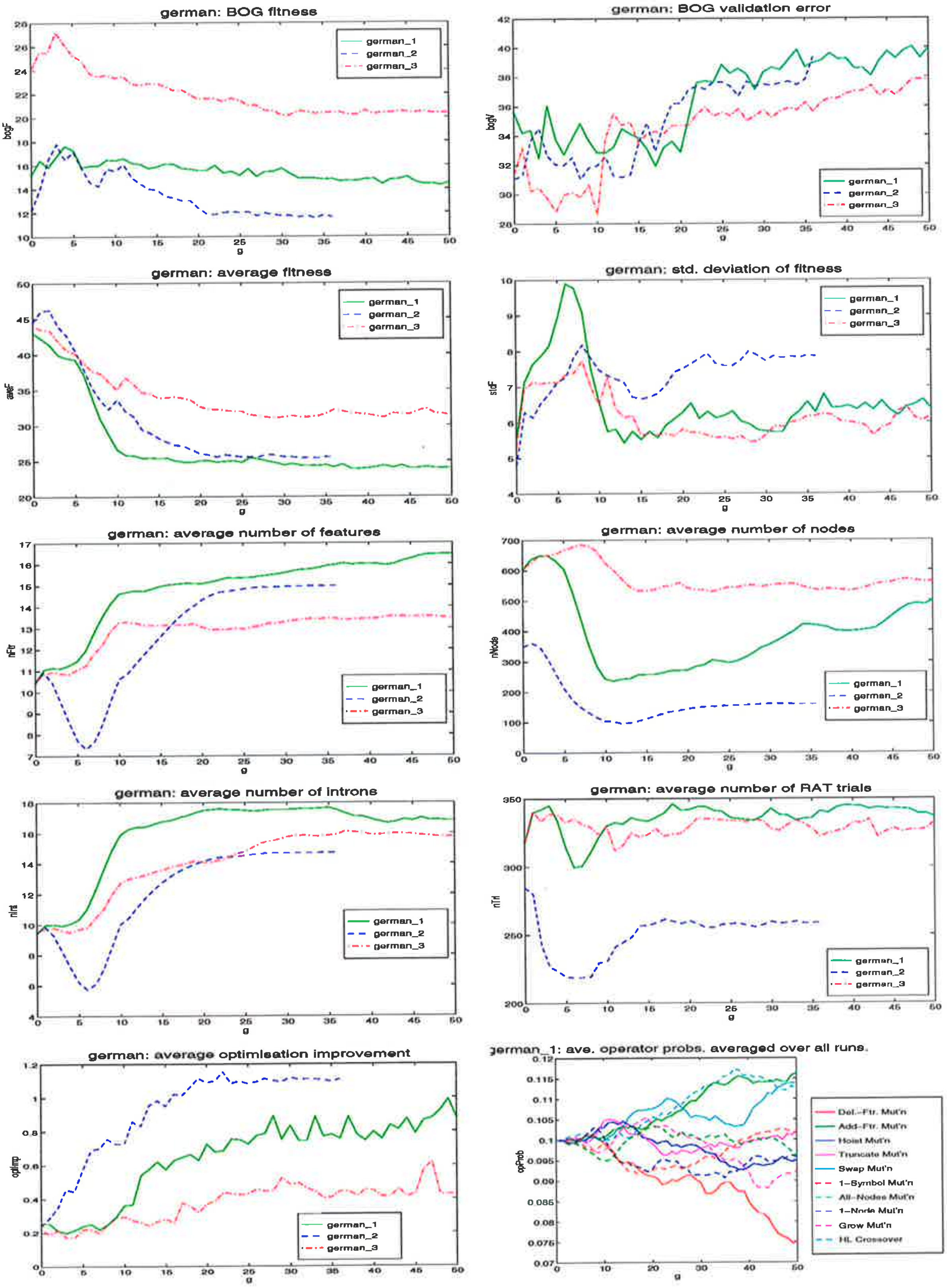


Figure E.7: Performance measures for **german**.

E.8 monks

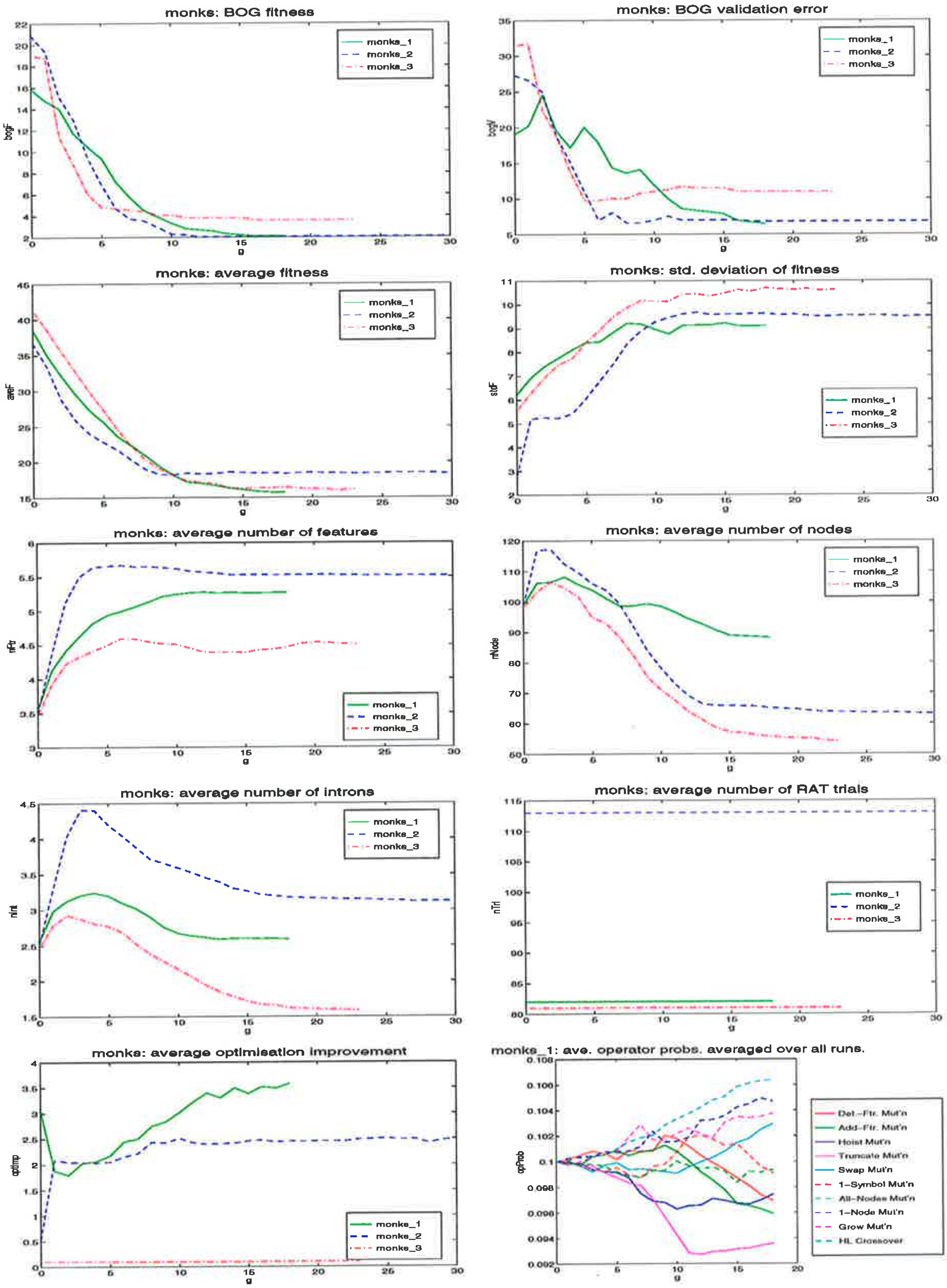


Figure E.8: Performance measures for **monks**.

E.9 satimage

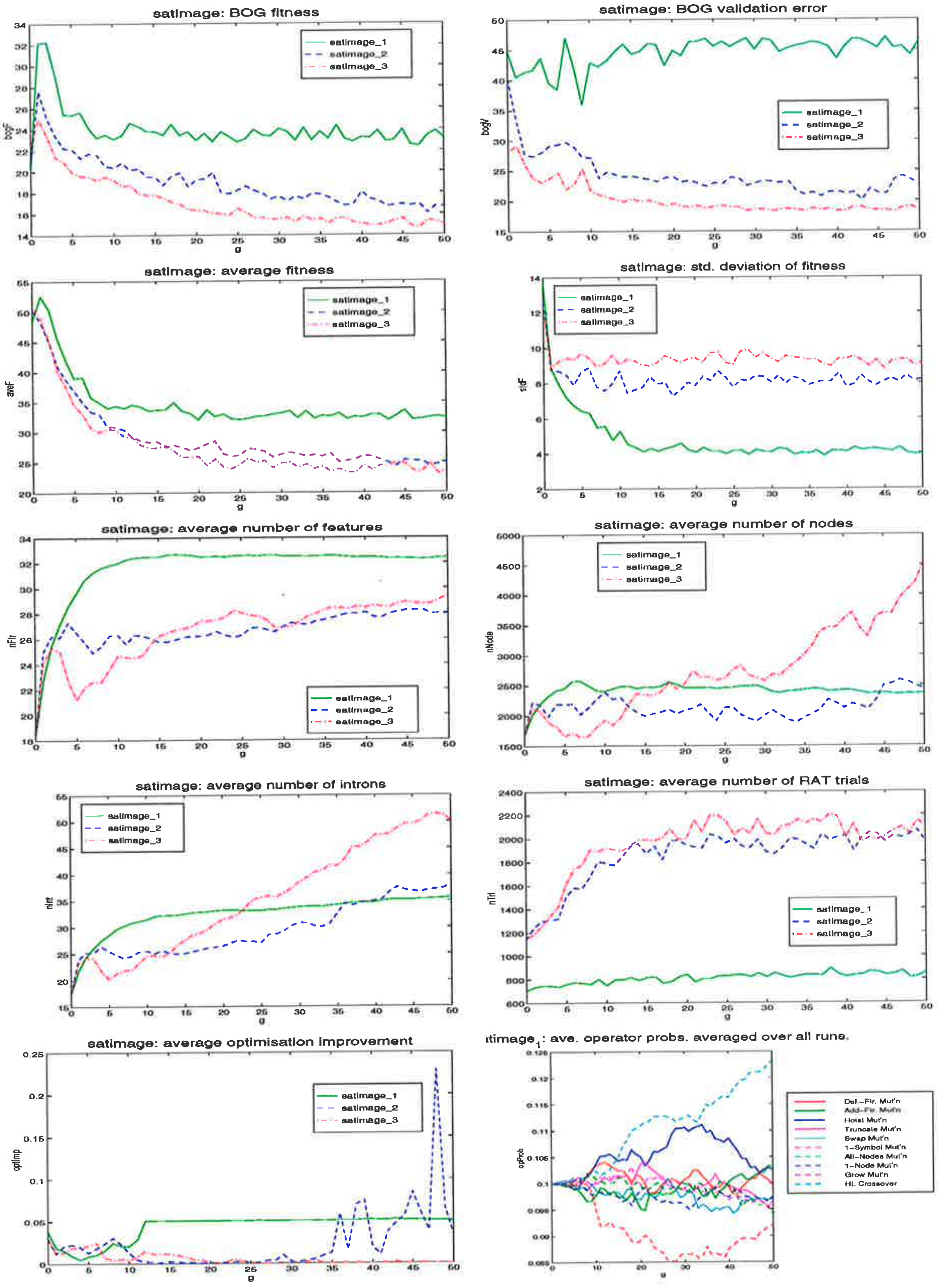


Figure E.9: Performance measures for **satimage**.

E.10 segment

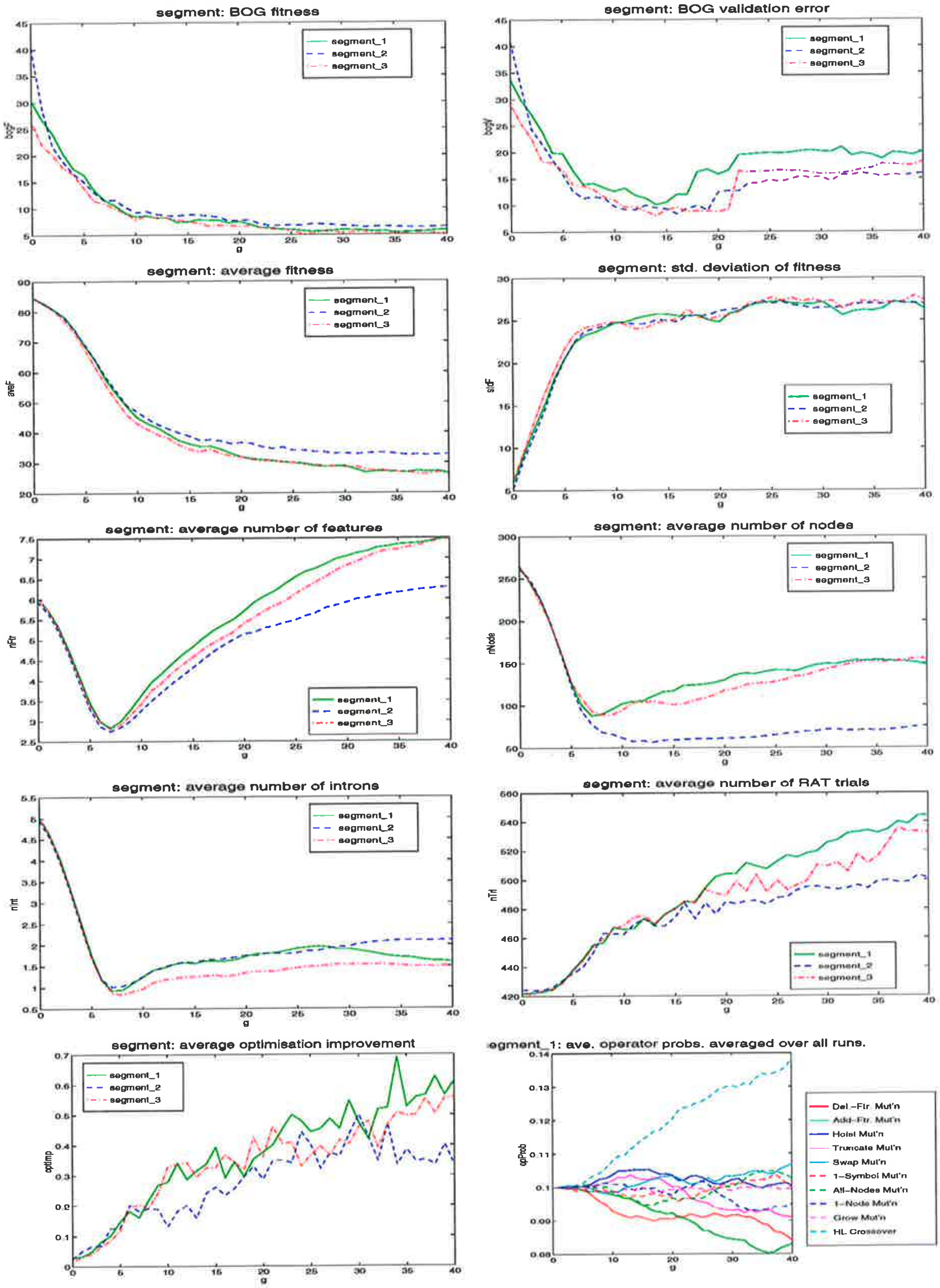


Figure E.10: Performance measures for **segment**.

E.11 smoking

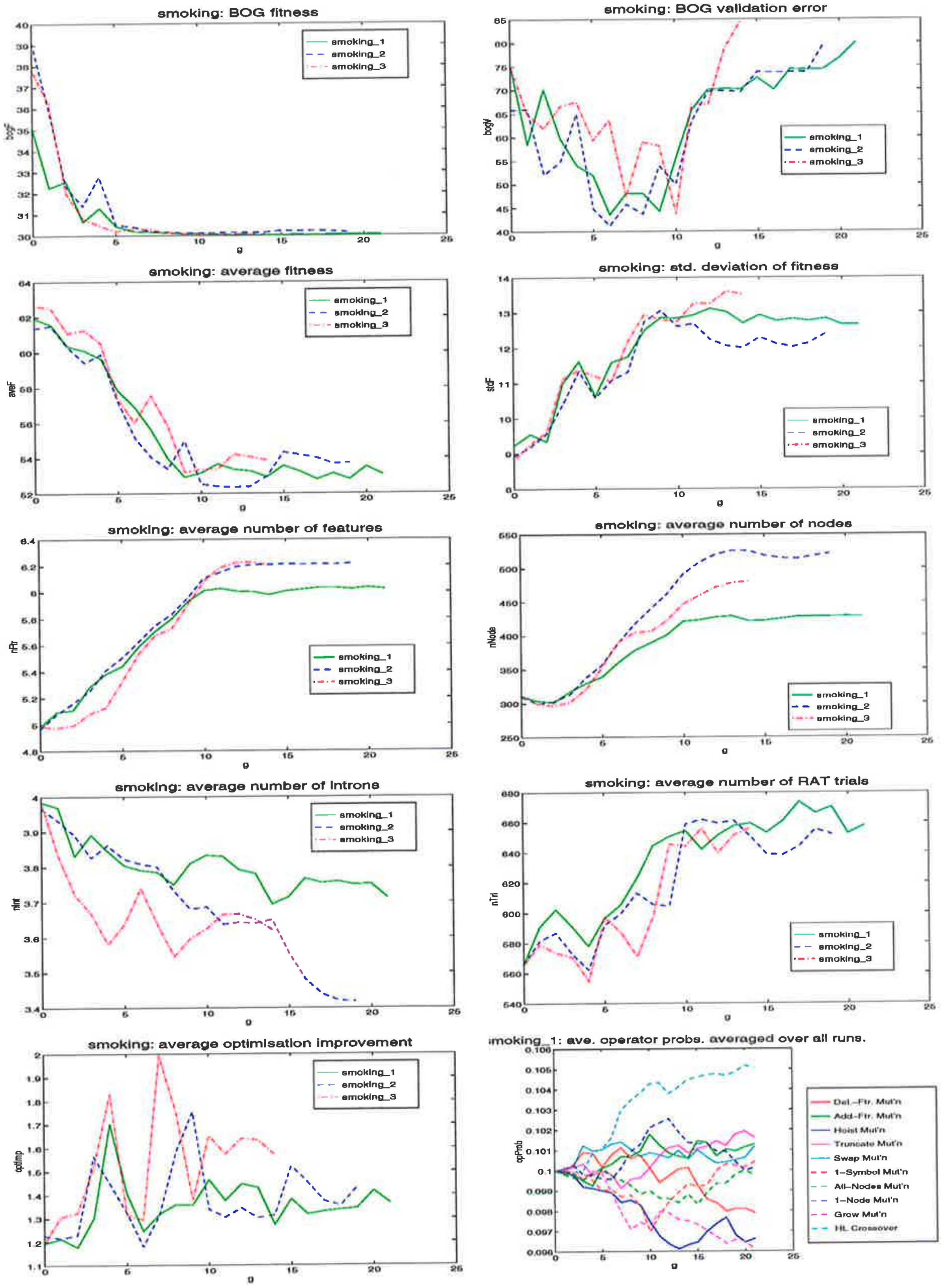


Figure E.11: Performance measures for **smoking**.

E.12 spirals

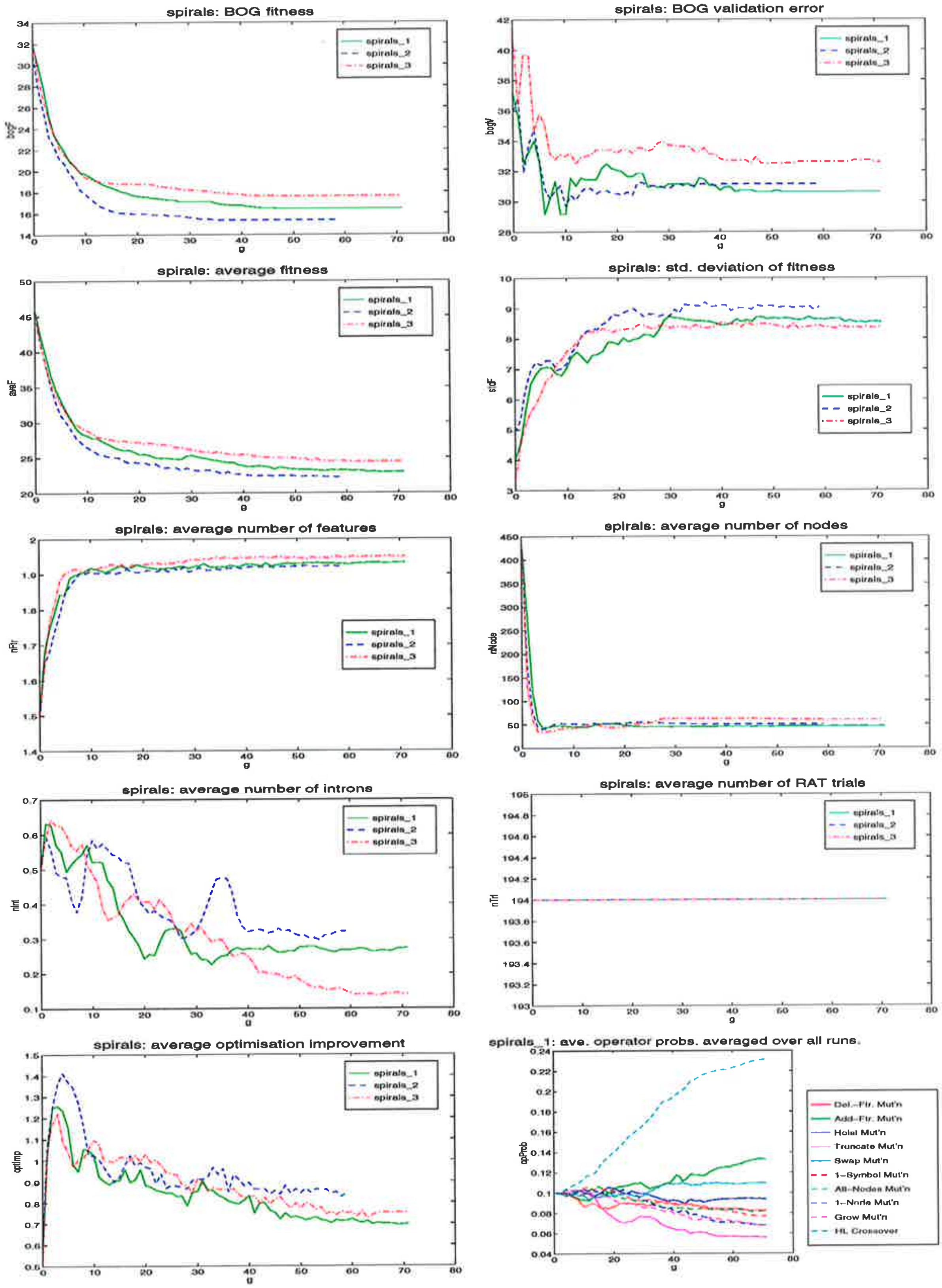


Figure E.12: Performance measures for **spirals**.

E.13 titanic

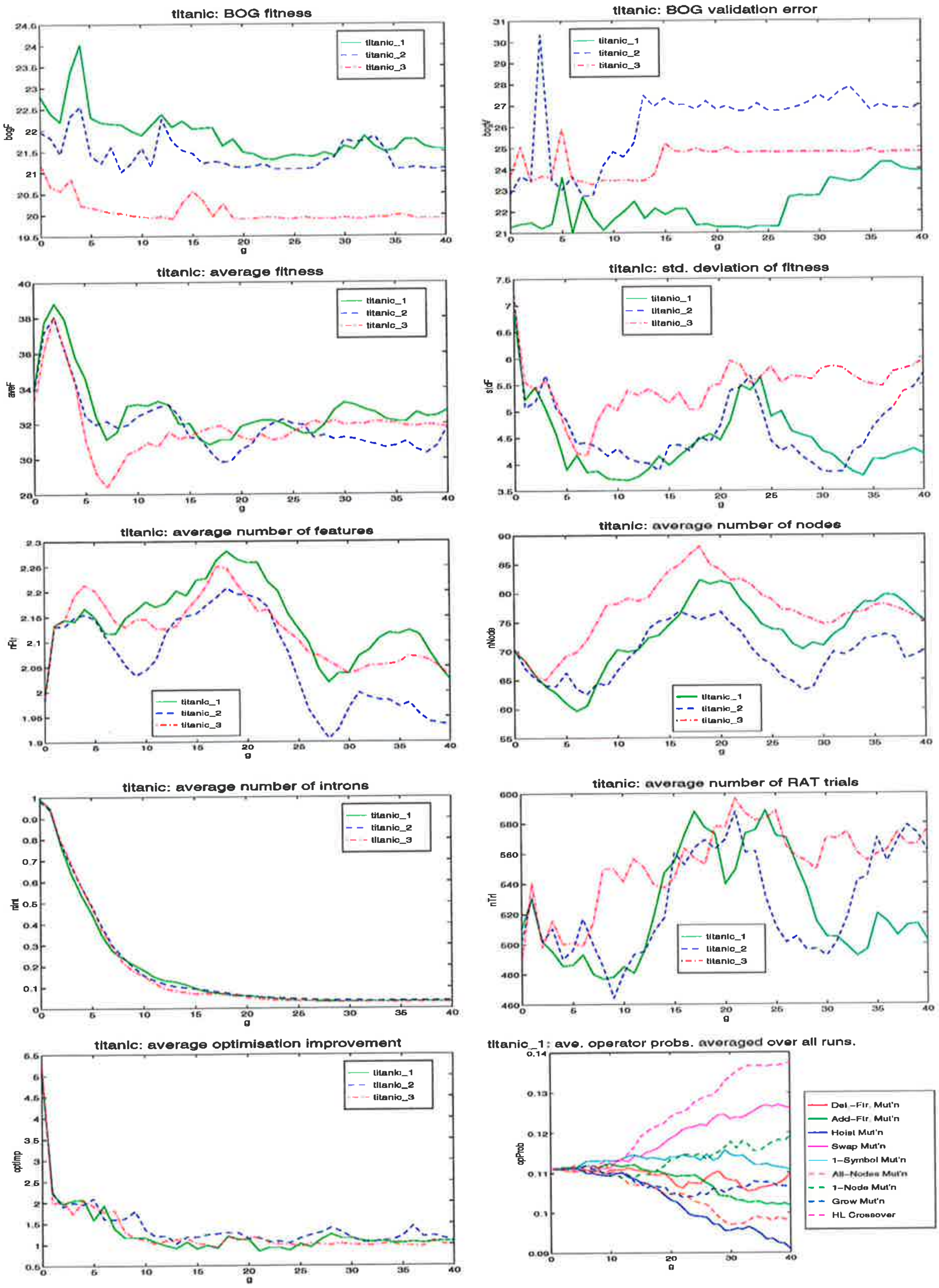


Figure E.13: Performance measures for **titanic**.

E.14 vehicle

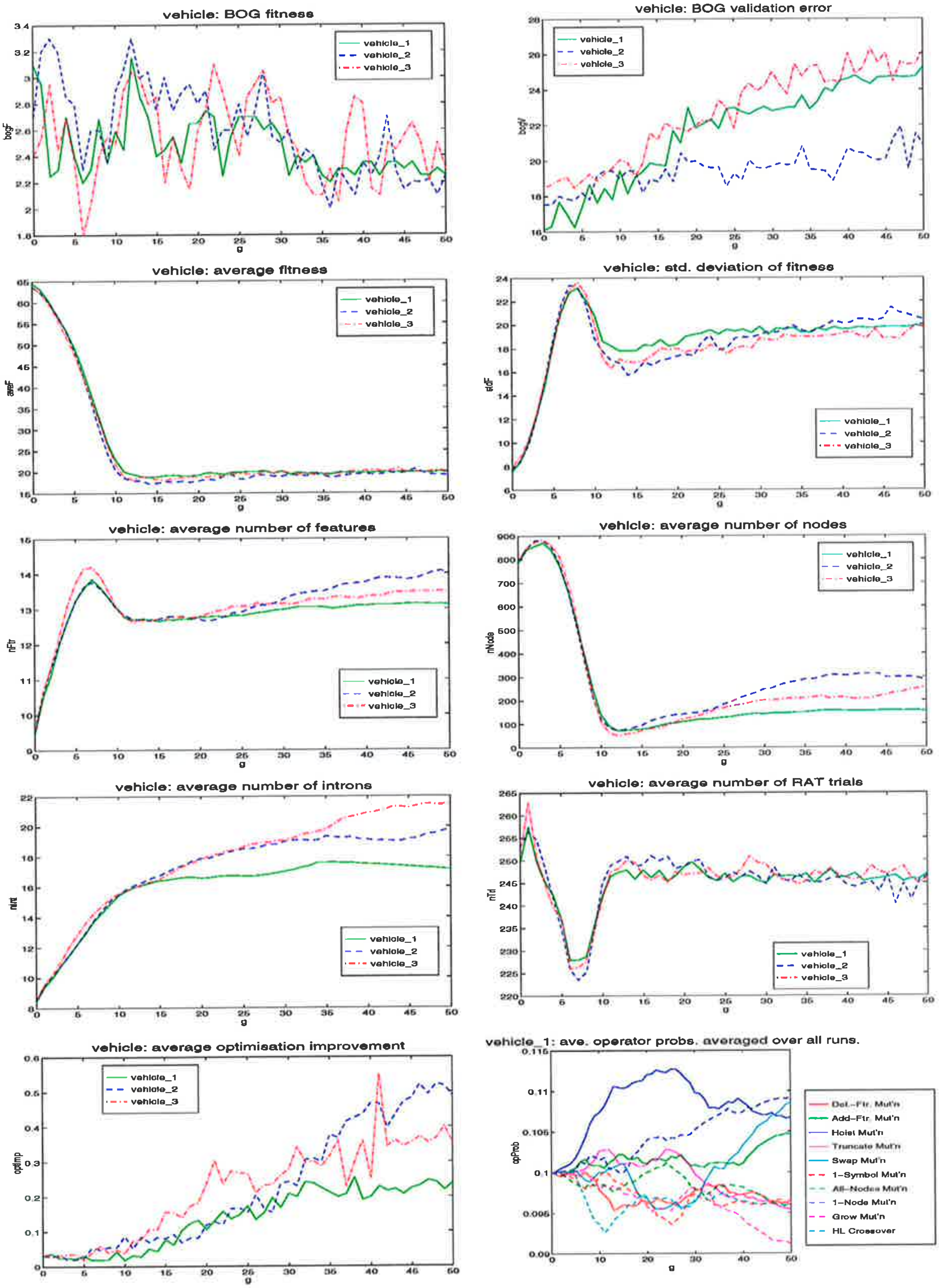


Figure E.14: Performance measures for **vehicle**.

E.15 yeast

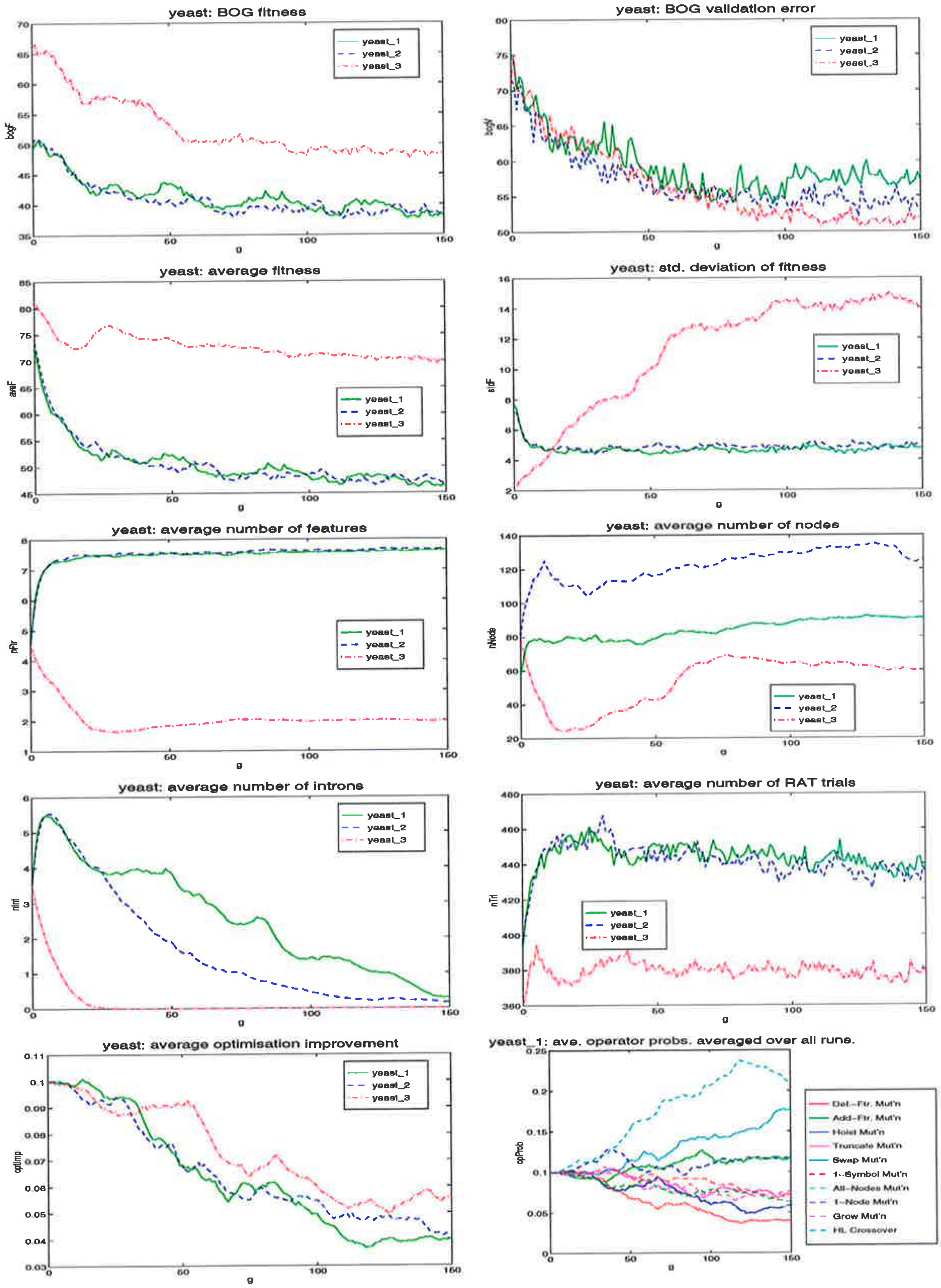


Figure E.15: Performance measures for **yeast**.

Bibliography

Altenberg, Lee. "The Evolution of Evolvability in Genetic Programming." In Kenneth E. Kinnear, Jr., ed., *Advances in Genetic Programming*, chap. 3, pp. 47–74. MIT Press, 1994.

Altenberg, Lee. "The Schema Theorem and Price's Theorem." In L. Darrell Whitley and Michael D. Vose, eds., *Foundations of Genetic Algorithms 3*, pp. 23–49. Estes Park, Colorado, USA: Morgan Kaufmann, 1995.

Altman, Russ B. "Molecular Biology for Computer Scientists Tutorial." Tutorial at Genetic Programming 1997 Conference, 1997.

Andre, David. "Automatically Defined Features: The Simultaneous Evolution of 2-Dimensional Feature Detectors and an Algorithm for Using Them." In Kenneth E. Kinnear, Jr., ed., *Advances in Genetic Programming*, chap. 23, pp. 477–494. MIT Press, 1994.

Angeline, P. J. and Pollack, J. B. "Coevolving High-Level Representations." July Technical report 92-PA-COEVOLVE, Laboratory for Artificial Intelligence. The Ohio State University, 1993.

Angeline, Peter J. "Adaptive and Self-Adaptive Evolutionary Computations." In M. Palaniswami, Y Attikiouzel, R. Marks, D. Fogel, and T. Fukuda, eds., *Computational Intelligence: A Dynamic Systems Perspective*, pp. 152–163. Piscataway, NJ: IEEE Press, 1995.

Angeline, Peter J. "Two Self-Adaptive Crossover Operators for Genetic Programming." In Peter J. Angeline and K. E. Kinnear, Jr., eds., *Advances in Genetic Programming 2*, chap. 5, pp. 89–110. Cambridge, MA, USA: MIT Press, 1996.

Angeline, Peter J. "An Alternative to Indexed Memory for Evolving Programs with Explicit State Representations." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 423–430. Stanford University, CA, USA: Morgan Kaufmann, 1997*a*.

Angeline, Peter J. "Subtree Crossover: Building Block Engine or Macromutation?" In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 9–17. Stanford University, CA, USA: Morgan Kaufmann, 1997*b*.

Angeline, Peter John. "Genetic Programming and Emergent Intelligence." In Kenneth E. Kinnear, Jr., ed., *Advances in Genetic Programming*, chap. 4, pp. 75–98. MIT Press, 1994.

Antonisse, Jim. "A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint." In J. D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 86–91. San Mateo, California USA: Morgan Kaufmann, 1989.

Aviles-Cruz, C., Guérin-Dugué, A., Van Cappel, D., and Voz, J.L. *Enhanced Learning for Evolutive Neural Architecture.* ESPRIT, 1995.

Bäck, Thomas. "Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms." In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 57–62. Piscataway NJ: IEEE Press, 1994.

Bäck, Thomas and Schwefel, Hans-Paul. "An Overview of Evolutionary Algorithms for Parameter Optimization." *Evolutionary Computation*, vol. 1(1), pp. 1–23, 1993.

Backer, Gerriet. "Learning with missing data using Genetic Programming." In *The 1st Online Workshop on Soft Computing (WSC1).* http://www.bioele.nuee.nagoya-u.ac.jp/wsc1/: Nagoya University, Japan, 1996.

Balakrishnan, Karthik and Honavar, Vasant. "Evolutionary Design of Neural Architectures - A Preliminary Taxonomy and Guide to Literature." TR CS 95-01, Artificial Intelligence Group, Iowa State University, Ames, Iowa 50011-1040 USA, 1995*a*.

Balakrishnan, Karthik and Honavar, Vasant. "Properties of Genetic Representations of Neural Architectures." *Proceedings of the World Congress on Neural Networks*, vol. 1, pp. 807–813, 1995*b*.

Baldwin, J. M. "A New Factor in Evolution." *American Naturalist*, vol. 30, pp. 441–451, 1896.

Banzhaf, Wolfgang, Frankone, Frank D., and Nordin, Peter. "The Effect of Extensive Use of the Mutation Operator on Generalization in Genetic Programming Using Sparse Data Sets." In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, eds., *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation*, vol. 1141 of *LNCS*, pp. 300–309. Berlin, Germany: Springer Verlag, 1996.

Bartlett, Eric B. "Dynamic Node Architecture Learning: An Information Theoretic Approach." *Neural Networks*, vol. 7(1), pp. 129–140, 1994.

Baum, E. B. and Haussler, D. "What Size Net Gives Valid Generalisation?" *Neural Computation*, vol. 1, pp. 151–160, 1989.

Berry, Michael J. A. and Linoff, Gordon. *Data Mining Techniques for marketing, sales, and customer support.* John Wiley and Sons, 1997.

Bishop, Christopher M. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

Blickle, Tobias. "Evolving Compact Solutions in Genetic Programming: A Case Study." Tik-report, TIK Institut fur Technische Informatik und Kommunikationsnetze, Computer Engineering and Networks Laboratory, ETH, Swiss Federal Institute of Technology, Gloriastrasse 35, 8092 Zurich, Switzerland, 1996*a*.

Blickle, Tobias. *Theory of Evolutionary Algorithms and Application to System Synthesis.* Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, 1996*b*.

Blickle, Tobias and Thiele, Lothar. "Genetic Programming and Redundancy." In J. Hopf, ed., *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, pp. 33–38. Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany: Max-Planck-Institut für Informatik (MPI-I-94-241), 1994.

Blösch, Anthony. "Treetex 1.0 - Software for the Automatic Layout of n-ary Trees." 1993.

Blum, A. and Rivest, R. "Training a 3-node neural netowrk is NP-complete." In *Proceedings of the 1988 Workshop on Computational Leraning Theory*, pp. 9–18. Boston, MA: Morgan Kaufmann, 1988.

Bornholdt, Stefan and Graudenz, Dirk. "General Asymmetric Neural Networks and Structure Design by Genetic Algorithms." *Neural Networks*, vol. 5, pp. 327–334, 1992.

Box, George E. P., Hunter, William G., and Hunter, J. Stuart. *Statistics for Experimenters.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 1978.

Branke, Jurgen. "Evolutionary Algorithms for Neural Network Design and Training." In Jarmo T Alander, ed., *Proceedings of the First Nordic Workshop on Genetic Algorithms and its Applications.* Vaasa, Finland, 1995.

Breiman, Leo, Friedman, Jerome H., Olshen, Richard A., and Stone, Charles J. *Classification and Regression Trees.* Wadsworth, 1984.

Bruce, Wilker Shane. "Automatic Generation of Object-Oriented Programs Using Genetic Programming." In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 267–272. Stanford University, CA, USA: MIT Press, 1996.

Brüggemann-Klein, Anne and Wood, Derick. "Drawin Trees Nicely with TEX." Tech. rep., Freiburg University and University of Waterloo, 1989.

Bull, Shelley. *Case Studies in Biometry*, chap. 13, pp. 249–271. Probability and Mathematical Statistics. New York, NY: John Wiley & Sons, 1994.

Calabretta, Farraele, Galbiati, Riccardo, Nolfi, Stefano, and Parisi, Domenico. "Investigating the Role of Diploidy in Simulated Populations of Evolving Individuals.", 1997. Submitted for publication.

Casasent, David P. and Neiberg, Leonard. "Distortion-Invariant Image Pattern Recognition FST Temporal Neural Network." *Proceedings of the World Congress on Neural Networks*, vol. 2, pp. 145–150, 1995.

Chellapilla, Kumar. "Evolutionary Programming with Tree Mutations: Evolving Computer Programs without Crossover." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 431–438. Stanford University, CA, USA: Morgan Kaufmann, 1997.

Chen, Yan Qiu, Thomas, David W., and Nixon, Mark S. "Generating-Shrinking Algorithm for Learning Arbitrary Classification." *Neural Networks*, vol. 7(9), pp. 1477–1489, 1994.

Colby, Chris. "Introduction to Evolutionary Biology." http://www.talkorigins.org/faqs/faq-intro-to-biology.html, 1996.

Demiral, Hasan Tahsin, Ma, Sheng, and Ji, Chuanyi. "Combined Power of Weak Classifiers." *Proceedings of the World Congress on Neural Networks*, vol. 1, pp. 591–595, 1995.

Droste, Stefan. "Efficient Genetic Programming for Finding Good Generalizing Boolean Functions." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 82–87. Stanford University, CA, USA: Morgan Kaufmann, 1997.

Duda, R.O. and Hart, P.E. *Pattern Classification and Scene Analysis.* Wiley-International, 1973.

Dunstone, Edward S. "Face Normalisation and Recognition using Low Frequency Gabor Jets and Neural Networks." *Proceedings of the Australian Conference on Neural Networks*, pp. 95–97, 1995.

Efron, Bradley and Tibshirani, Robert J. *An Introduction to the Bootstrap.* No. 57 in Monographs on Statistics and Applied Probability. Chapman & Hall, 1993.

Ellison, Mark. "William of Ockham." In *Minimum Message Length Encoding.* http://www.cs.monash.edu.au/-~lloyd/tildeMML/Notes/Ockham.html: Department of Computer Science, Monash University, 1995.

Esparcia-Alcazar, Anna I. and Sharman, Ken. "Evolving Recurrent Neural Network Architectures by Genetic Programming." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 89–94. Stanford University, CA, USA: Morgan Kaufmann, 1997.

ESPRIT. "Project StatLog." 1995. Datasets and Software Available from LIACC, Porto via ftp://ftp.ncc.up.pt/pub/statlog.

Feelders, A. and Verkooijen, W. "Which Method Learns Most From the Data?" Tech. rep., University of Twente, Department of Computer Science, the Netherlands, 1995. Anonymous FTP: /pub/doc/pareto/aistats95.ps.Z on ftp.cs.utwente.nl.

Fiesler, E. "Minimal and High Order Neural network Topologies." In *Proceedings of the 1993 international Simulation Technology conference (SimTec '93).* Society for Computer Simulation (SCS), San Diego, 1993.

Flanagan, David. *Java in a Nutshell - a Desktop Quick Reference.* The Java Series. O'Reilly, 2nd edn., 1997.

Flexer, Arthur. "Statistical Evaluation of Neural Network Experiments: Minimum Requirements and Current Practice." In Trappl R., ed., *Cybernetics and Systems '96, Proceedings of the 13th European Meeting on Cybernetics and Systems Research*, vol. 2, pp. 1005–1008. Austrian Society for Cybernetic Studies, Vienna, 1996.

Fogel, David B. *Evolutionary Computation - Toward a New Philosophy of Machine Intelligence.* IEEE Press, 1995.

Fogel, David B. "Evolutionary Programming and Evolution Strategies Tutorial." Tutorial at Genetic Programming 1997 Conference, 1997.

Fonseca, Carlos M. and Fleming, Peter J. "An Overview of Evolutionary Algorithms in Multiobjective Optimization." *Evolutionary Computation*, vol. 3(1), pp. 1–16, 1995.

Foulds, L. R. *Optimization Techniques: an Introduction.* Springer-Verlag, 1981.

Francone, Frank D., Nordin, Peter, and Banzhaf, Wolfgang. "Benchmarking the Generalization Capabilities of a Compiling Genetic programming System using Sparse Data Sets." In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 72–80. Stanford University, CA, USA: MIT Press, 1996.

Friedman, H.P. and Rubin, J. "On Some Invariant Criteria for Grouping Data." *American Statistical Association Journal*, vol. 62, pp. 1159–1178, 1967.

Friedman, Jerome H. "Local Learning Based on Recursive Covering." Tech. rep., Department of Statistics, Stanford University, 1996.

Friedrich, Christoph M. and Moraga, Claudio. "An Evolutionary Method to Find Good Building-Blocks for Architectures of Artificial Neural Networks." In *Proceedings of the Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU '96)*, pp. 951–956. Granada, Spain, 1996.

Fritzke, Bernd. "Incremental Learning of Local Linear Mappings." *Proceedings of the International Conference on Artificial Neural Networks*, 1995.

Fröhlich, Jürg and Hafner, Christian. "Extended and Generalized Genetic Programming for Function Analysis." Tech. rep., Swiss Federal Institute of Technology, Laboratory for Electromagnetic Fields and Microwaves Electronics. Available from http://alphard.ethz.ch/hafner/gp.htm, 1996.

Fu, K. S. *Digital Pattern Recognition.* No. 10 in Communication and Cybernetics. Springer-Verlag, 2nd edn., 1980.

Fukunaga, Keinosuke. *Introduction to Statistical Pattern Recognition.* Academic Press, 2nd edn., 1990.

Gaarder, Jostein. *Sophie's World.* Phoenix House, 1996.

Gallinari, P., Thiria, S., Badran, F., and Fogelman-Soulie, F. "On the Relations Between Discriminant Analysis and Multilayer Perceptrons." *Neural Networks*, vol. 4, pp. 349–360, 1991.

Garey, Michael R. and Johnson, David S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman San Fransisco, 1979.

Gathercole, Chris and Ross, Peter. "Small Populations over Many Generations can beat Large Populations over Few Generations in Genetic Programming." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 111–118. Stanford University, CA, USA: Morgan Kaufmann, 1997*a*.

Gathercole, Chris and Ross, Peter. "Tackling the Boolean Even N Parity Problem with Genetic Programming and Limited-Error Fitness." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 119–127. Stanford University, CA, USA: Morgan Kaufmann, 1997*b*.

Gelfand, Saul B. and Delp, Edward J. "On Tree Structured Classifiers." In I. K. Sethi and A. K. Jain, eds., *Artificial Neural Networks and Statistical Pattern Recognition - Old and New Connections*, pp. 51–87. Elsevier Science Publishers, 1991.

Goldberg, David E., Deb, Kalyanmoy, and Clark, James H. "Genetic Algorithms, noise, and the sizing of populations." IlliGAL Report 91010, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 1991.

Goldberg, D.E. *Genetic Algorithms in Search, Optimisation, and Machine Learning.* Addison-Wesley Publishing Company, Inc., 1989.

Goldschlager, Les and Lister, Andrew. *Computer Science: a Modern Introduction.* Computer Science. Prentice Hall International, 2nd edn., 1988.

Gower, J.C. and Ross, G.J.S. "Minimum Spanning Trees and Single Linkage Cluster Analysis." *Applied Statistics*, vol. 18(1), pp. 54–64, 1969.

Gray, Charles M., König, Peter, Engel, Andreas K., and Singer, Wolf. "Oscillatory Responses in Cat Visual Cortex Exhibit Inter-columnar Synchronization which Reflects Global Stimulus Properties." *Nature*, vol. 338, pp. 334–337, 1989.

Gray, H. F., Maxwell, R. J., Martinez-Perez, I., Arus, C., and Cerdan, S. "Genetic Programming for Classification of Brain Tumours from Nuclear Magnetic Resonance Biopsy Spectra." In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, p. 424. Stanford University, CA, USA: MIT Press, 1996.

Gray, Helen. "Genetic Programming for Classification of Medical Data." In John R. Koza, ed., *Late Breaking Papers at the 1997 Genetic Programming Conference*, p. 291. Stanford University, CA, USA: Stanford Bookstore, 1997.

Grefenstette, J. J. "Conditions for implicit parallelism." In G. J. E. Rawlins, ed., *Foundations of Genetic Algorithms*, pp. 252–261. San Mateo, CA: Morgan Kaufmann, 1991.

Grefenstette, John J. and Baker, James E. "How Genetic Algorithms Work: a Critical Look at Implicit Parallelism." In J. D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 20–27. San Mateo, California USA: Morgan Kaufmann, 1989.

Gruau, F. "Cellular encoding of Genetic Neural Networks." Technical report 92-21, Laboratoire de l'Informatique du Parallilisme. Ecole Normale Supirieure de Lyon, France, 1992.

Guo, Heng and Gelfand, Saul B. "Classification Trees with Neural Network Feature Extraction." *IEEE Transactions on Neural Networks*, vol. 3(6), pp. 923–933, 1992.

Hafner, Christian, Froehlich, Juerg, and Gerber, Hansueli. "Generalized Genetic Program.", 1996. Submitted to the *Evolutionary Computation* Journal.

Handley, Simon. "Classifying Nucleic Acid Sub-Sequences as Introns or Exons Using Genetic Programming." In Christopher Rawlins, Dominic Clark, Russ Altman, Lawrence Hunter, Thomas Lengauer, and Shoshana Wodak, eds., *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology (ISMB-95)*, pp. 162–169. Cambridge, UK: AAAI Press, 1995.

Hanson, Robin, Stutz, John, and Cheeseman, Peter. "Bayesian Classification Theory." Tech. Rep. FIA-90-12-7-01, Artificial Intelligence Research Branch, NASA Ames Research Centre, CA USA, 1991.

Harries, Kim and Smith, Peter. "Exploring Alternative Operators and Search Strategies in Genetic Programming." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 147–155. Stanford University, CA, USA: Morgan Kaufmann, 1997.

Haykin, Simon. *Neural Networks: A Comprehensive Foundation.* Macmillan College Publishing Company, 1994.

Haynes, Thomas. "Phenotypical Building Blocks for Genetic Programming." In Eric Goodman, ed., *Genetic Algorithms: Proceedings of the Seventh International Conference.* Michigan State University, East Lansing, MI, USA: Morgan Kaufmann, 1997.

Hiden, Hugo, Willis, Mark, McKay, Ben, and Montague, Gary. "Non-Linear And Direction Dependent Dynamic Modelling Using Genetic Programming." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 168–173. Stanford University, CA, USA: Morgan Kaufmann, 1997.

Hinterding, R., Michalewicz, Z., and Eiben, A. E. "Adaptation in Evolutionary Computation: a Survey." In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pp. 65–69. IEEE Service Centre, 1997.

Holland, John H. *Adaptation in Natural and Artificial Systems.* MIT Press, 1995.

Horn, Jeffrey and Nafpliotis, Nicholas. "Multiobjective Optimization Using the Niched Pareto Genetic Algorithm." IlliGAL Report 93005, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois at Urbana-Champaign, 1993.

Houck, C. R., Joines, J. A., and Kay, M. G. "Utilizing Lamarckian evolution and the Baldwin effect in hybrid genetic algorithms." Tech. Rep. NCSU-IE 96-01, College of Engineering, North Carolina State University, 1996.

Iba, Hitoshi, de Garis, Hugo, and Sato, Taisuke. "Genetic Programming Using a Minimum Description Length Principle." In Kenneth E. Kinnear, Jr., ed., *Advances in Genetic Programming*, chap. 12, pp. 265–284. MIT Press, 1994*a*.

Iba, Hitoshi, de Garis, Hugo, and Sato, Taisuke. "Genetic Programming with Local Hill-Climbing." In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, eds., *Parallel Problem Solving from Nature III*, pp. 334–343. Jerusalem: Springer-Verlag, 1994*b*.

Iba, Hitoshi, Karita, Takio, de Garis, Hugo, and Sato, Taisuke. "System Identification Using Structured Genetic Algorithms." In Stephanie Forrest, ed., *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pp. 279–286. University of Illinois at Urbana-Champaign: Morgan Kaufmann, 1993*a*.

Iba, Hitoshi, Kurita, Takio, de Garis, Hugo, and Sato, Taisuke. "System Identification using Structured Genetic Algorithms." In Stephanie Forrest, ed., *Proceedings of the Fifth International Conference on Genetic Algorithms.* University of Illinois at Urbana-Champaign, Morgan Kaufmann Publishers, 1993*b*.

James, G. and Hastie, T. "Generalizations of the Bias/Variance Decomposition for Prediction Error." Tech. rep., Department of Statistics, Stanford University, 1997.

Jaske, Harri. "On code reuse in genetic programming." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 201–206. Stanford University, CA, USA: Morgan Kaufmann, 1997.

John, George H., Kohavi, Ron, and Pfleger, Karl. "Irrelevant Features and the Subset Selection problem." In William W. Cohen and Haym Hirsh, eds., *Machine learning : proceedings of the eleventh international conference*, pp. 121–129. New Brunswick, N.J.: Rutgers University, 1994.

Jong, Kenneth A. De and Sarma, Jayshree. "Generation Gaps Revisited." In Darrell Whitley, ed., *Foundations of Genetic Algorithms - 2*, pp. 19–28. Morgan Kaufmann, 1992.

K. Chellapilla, David B. Fogel. "Exploring Self-Adaptive Methods to Improve the Efficiency of Generating Approximate Solutions to Traveling Salesman Problems Using Evolutionary Programming." In *Evolutionary Programing 97.* Indianapolis, IN, 1997.

Keith, Mike J. and Martin, Martin C. "Genetic Programming in C++: Implementation Issues." In Kenneth E. Kinnear, Jr., ed., *Advances in Genetic Programming*, chap. 13, pp. 285–310. MIT Press, 1994.

Kinnear, Jr., Kenneth E. "Evolving a Sort: Lessons in Genetic Programming." In *Proceedings of the 1993 International Conference on Neural Networks*, vol. 2. San Francisco, USA: IEEE Press, 1993*a*.

Kinnear, Jr., Kenneth E. "Generality and Difficulty in Genetic Programming: Evolving a Sort." In Stephanie Forrest, ed., *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pp. 287–294. University of Illinois at Urbana-Champaign: Morgan Kaufmann, 1993*b*.

Kinnear, Jr., Kenneth E. "A perspective on the Work in this Book." In Kenneth E. Kinnear, Jr., ed., *Advances in Genetic Programming*, chap. 1, pp. 3–19. MIT Press, 1994.

Kopka, Helmut and Daly, Patrick W. *A Guide to LaTeX 2ε - Document Preparation for Beginners and Advanced Users.* Addison-Wesley, 2nd edn., 1995.

Koza, John R. "A Genetic Approach to the Truck Backer Upper Problem and the Inter-Twined Spiral Problem." In *Proceedings of IJCNN International Joint Conference on Neural Networks*, vol. IV, pp. 310–318. IEEE Press, 1992*a*.

Koza, John R. *Genetic Programming: On the Programming of Computers by Natural Selection.* Cambridge, MA, USA: MIT Press, 1992*b*.

Koza, John R. "Automated discovery of detectors and iteration-performing calculations to recognize patterns in protein sequences using genetic Programming." In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pp. 684–689. IEEE Computer Society Press, 1994*a*.

Koza, John R. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge Massachusetts: MIT Press, 1994*b*.

Koza, John R., Andre, David, Bennett III, Forrest H., and Keane, Martin A. "Use of Automatically Defined Functions and Architecture-Altering Operations in Automated Circuit Synthesis Using Genetic Programming." In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 132–149. Stanford University, CA, USA: MIT Press, 1996*a*.

Koza, John R., Bennett III, Forest H., Lohn, Jason, Dunlap, Frank, Keane, Martin A., and Andre, David. "Use of Architecture-Altering Operations to Dynamically Adapt a Three-Way Analog Source Identification Circuit to Accommodate a New Source." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 213–221. Stanford University, CA, USA: Morgan Kaufmann, 1997.

Koza, John R., Bennett III, Forrest H., Andre, David, and Keane, Martin A. "Four problems for which a computer program evolved by genetic programming is competitive with human performance." In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, vol. 1, pp. 1–10. IEEE Press, 1996*b*.

Kreinovich, Vladik Ya. "Arbitrary Nonlinearity Is Sufficient to Represent All Functions by Neural Networks: A Theorem." *Neural Networks*, vol. 4, pp. 381–383, 1991.

Kruizinga, P. and Petkov, N. "Person Identification Based on Multiscale Matching of Cortical Images." *Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*, 1995.

Lakos, John. *Large-scale C++ Software Design.* Professional Computing Series. Addison-Wesley, 1996.

Lang, Kevin J. and Whitbrock, Michael J. "Learning to Tell Two Spirals Apart." In David S. Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski, eds., *Proceedings of the 1988 Connectionist Models Summer School.* Morgan Kaufmann, 1989.

Langdon, W. B. "Pareto, Population Partitioning, Price and Genetic Programming." Research Note RN/95/29, University College London, Gower Street, London WC1E 6BT, UK, 1995.

Langdon, W. B. *Data Structures and Genetic Programming.* Ph.D. thesis, University College, London, 1996*a*.

Langdon, W. B. and Poli, R. "An Analysis of the MAX Problem in Genetic Programming." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 222–230. Stanford University, CA, USA: Morgan Kaufmann, 1997*a*.

Langdon, W. B. and Poli, R. "Fitness Causes Bloat." Tech. Rep. CSRP-97-09, University of Birmingham, School of Computer Science, Birmingham, B15 2TT, UK, 1997*b*.

Langdon, William B. "Data Structures and Genetic Programming." In Peter J. Angeline and K. E. Kinnear, Jr., eds., *Advances in Genetic Programming 2*, chap. 20, pp. 395–414. Cambridge, MA, USA: MIT Press, 1996*b*.

Lawrence, Steve, Giles, C. Lee, Tsoi, Ah Chung, and Back, Andrew D. "Face Recognition: A Hybrid Neural Network Approach." Tech. Rep. UMIACS-TR-96-16, Institute for Advanced Computer Studies, University of Maryland, 1996.

Levenick, James R. "Inserting Introns Improves Genetic Algorithm Success Rate: Taking a Cue from Biology." In *Proceedings of the fourth International Conference on Genetic Algorithms*, pp. 123–127. 1991.

Lillesand, Thomas M. and Kiefer, Ralph W. *Remote Sensing and Image Interpretation.* John Wiley & Sons, Inc., 3rd edn., 1994.

Lim, Tjen-Sien, Loh, Wei-Yin, and Shih, Yu-Shan. "An Empirical Comparison of Decision Trees and Other Classification Methods." Tech. Rep. 979, Department of Statistics, University of Wisconsin, Madison, USA, 1997.

Loh, W.-Y. and Vanichsetakul, N. "Tree-structured Classification via Generalized Discriminant Analysis (with discussion)." *Journal of the Americal Statistical Association*, vol. 83, pp. 715–728, 1988.

Loh, Wei-Yin and Shih, Yu-Shan. "QUEST (Quick, Unbiased and Efficient Statistical Tree), version 1.6." http://www.stat.wisc.edu/~loh/quest.html, 1997*a*.

Loh, Wei-Yin and Shih, Yu-Shan. "Split Selection Methods for Classification Trees." *Statistica Sinica*, vol. 7, pp. 815–840, 1997*b*.

Luke, Sean and Spector, Lee. "Evolving Teamwork and Coordination with Genetic Programming." In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 150–156. Stanford University, CA, USA: MIT Press, 1996.

Luke, Sean and Spector, Lee. "A Comparison of Crossover and Mutation in Genetic Programming." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 240–248. Stanford University, CA, USA: Morgan Kaufmann, 1997.

Luttrell, S. P. "Using Self-Organising Maps to Classify Radar Range Profiles." *Proceedings of the IEE Conference on Artificial Neural Networks*, pp. 335–340, 1995.

Macready, William G. and Wolpert, David H. "On 2-armed Gaussian Bandits and Optimization." Tech. Rep. 96-03-009, Santa Fe Institute, 1996.

Marascuilo, Leonard A. and McSweeney, Maryellen. *Nonparametric and Distribution-Free Methods for the Social Sciences.* Wadsworth Publishing Company, Inc., 1977.

Mason, Andrew. "A Non-Linearity Measure of a Problem's Crossover Suitability." In *1995 IEEE Conference on Evolutionary Computation.* Perth, Australia: IEEE Press, 1995.

Meir, R. "Bias, Variance and the Combination of Least-Squares Estimators." In G. Tesauro, D. Touretzky, and T. Leen, eds., *Advnces in Neural Information Processing Systems 7*, pp. 295–302. MIT Press, 1994.

Merz, C.J. and Murphy, P.M. "UCI Repository of machine learning databases." 1996. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Meyer, Mike. "StatLib: a system for distributing statistical software, datasets, and information by electronic mail, FTP and WWW." http://lib.stat.cmu.edu/, 1996.

Microsystems, Sun. "XView: X Window-System-based Visual/Integrated Environment for Workstations." XView is available with the OpenWindows distribution, 1991.

Microsystems, Sun. "The Java Development Kit." http://java.sun.com, 1998.

Miller, Geoffrey F., Todd, Peter M., and Hegde, Shailesh U. "Designing Neural Networks using Genetic Algorithms." *Proceedings of the International Conference on Genetic Algorithms*, pp. 379–384, 1989.

Minsky, M. and Papert, S. *Perceptrons.* Cambridge: MIT Press, 1969.

Montana, David J. "Strongly Typed Genetic Programming." BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, 1993.

Moody, John and Darken, Christian J. "Fast Learning in Networks of Locally-Tuned Processing Units." *Neural Computation*, vol. 1, pp. 281–294, 1989.

Moore, Andrew W and Lee, Mary S. "Efficient Algorithms for Minimizing Cross Validation Error." In William W. Cohen and Haym Hirsh, eds., *Machine Learning: Proceedings of the Eleventh International Conference*, pp. 190–198. Rutgers University, Morgan Kaufmann, 1994.

Murthy, Kolluru Venkata Sreerama. *On Growing Better Decision Trees from Data.* Ph.D. thesis, John Hopkins University, Baltimore, Maryland, 1996.

Nelder, J. A. and Mead, R. "A Simplex Method for Function Minimization." *The Computer Journal*, vol. 7, pp. 308–313, 1965.

Nilsson, Nils J. *The Mathematical Foundations of Learning Machines.* Morgan Kaufmann Publishers, 1993.

Nordin, Peter and Banzhaf, Wolfgang. "Complexity Compression and Evolution." In L. Eshelman, ed., *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pp. 310–317. Pittsburgh, PA, USA: Morgan Kaufmann, 1995.

Nordin, Peter, Francone, Frank, and Banzhaf, Wolfgang. "Explicitly Defined Introns and Destructive Crossover in Genetic Programming." In Justinian P. Rosca, ed., *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pp. 6–22. Tahoe City, California, USA, 1995.

O'Reilly, U. M. and Oppacher, F. "The Troubling Aspects of a Building Block Hypothesis for Genetic Programming." Working Paper 94-02-001, Santa Fe Institute, 1399 Hyde Park Road Santa Fe, New Mexico 87501-8943 USA, 1992.

O'Reilly, Una-May and Oppacher, Franz. "Program Search with a Hierarchical Variable Length Representation: Genetic Programming, Simulated Annealing and Hill Climbing." Tech. Rep. 94-04-021, Santa Fe Institute, 1399 Hyde Park Road Santa Fe, New Mexico 87501-8943 USA, 1994*a*.

O'Reilly, Una-May and Oppacher, Franz. "Program Search with a Hierarchical Variable Length Representation: Genetic Programming, Simulated Annealing and Hill Climbing." In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner, eds., *Parallel Problem Solving from Nature – PPSN III*, no. 866 in Lecture Notes in Computer Science, pp. 397–406. Jerusalem: Springer-Verlag, 1994*b*.

O'Reilly, Una-May and Oppacher, Franz. "Using Building Block Functions to Investigate a Building Block Hypothesis for Genetic Programming." Working Paper 94-02-029, Santa Fe Institute, 1399 Hyde Park Road Santa Fe, New Mexico 87501-8943 USA, 1994*c*.

O'Reilly, Una-May and Oppacher, Franz. "Hybridized Crossover-Based Search Techniques for Program Discovery." Tech. Rep. 95-02-007, Santa Fe Institute, 1399 Hyde Park Road Santa Fe, New Mexico 87501-8943 USA, 1995*a*.

O'Reilly, Una-May and Oppacher, Franz. "The Troubling Aspects of a Building Block Hypothesis for Genetic Programming." In L. Darrell Whitley and Michael D. Vose, eds., *Foundations of Genetic Algorithms 3*, pp. 73–88. Estes Park, Colorado, USA: Morgan Kaufmann, 1995*b*.

O'Reilly, Una-May and Oppacher, Franz. "A Comparative Analysis of GP." In Peter J. Angeline and K. E. Kinnear, Jr., eds., *Advances in Genetic Programming 2*, chap. 2, pp. 23–44. Cambridge, MA, USA: MIT Press, 1996.

Parekh, Rajesh, Yang, Jihoon, and Honavar, Vasant. "Constructive Neural Network Learning Algorithms for Multi-Category Pattern Classification." Tech. Rep. 95-15, Artificial Intelligence Research Group, Iowa State University, Ames, Iowa 50011-1040 USA, 1995.

Peng, Fengchun, Jacobs, Robert A., and Tanner, Martin A. "Bayesian Inference in Mixtures-of-Experts and Hierarchical Mixtures-of-Experts Models With an Application to Speech Recognition.", 1995. Accepted for publication in the *Journal of the American Statistical Association*.

Perry, Chad. "How to write a Doctoral Thesis - PhD / DPhil.", 1998. To appear in the *Australasian Marketing Journal*, available on-line at http://www.imc.org.uk/imc/news/occpaper/cpindex.htm.

Plutowski, Mark and White, Halbert. "Active selection of training examples for network learning in noiseless environments." Tech. Rep. CS91-180, University of California, San Diego, 1991.

Poli, Riccardo and Langdon, W. B. "A New Schema Theory for Genetic Programming with One-point Crossover and Point Mutation." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 278–285. Stanford University, CA, USA: Morgan Kaufmann, 1997.

Pöppel, E. and Logothetis, N. "Neuronal Oscillations in the Human Brain." *Naturwissenschaften*, vol. 73, pp. 267–268, 1986.

Prechelt, Lutz. "PROBEN1 — A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms." Tech. Rep. 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, 1994. Anonymous FTP: /pub/papers/techreports/1994/1994-21.ps.Z on ftp.ira.uka.de.

Prechelt, Lutz. "A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice." *Neural Networks*, vol. 9, 1996.

Press, William H., Teukolsky, Saul A., Vetterling, William T., and Flannery, Brian P. *Numerical Recipes in C: the Art of Scientific Computing*. On-line at http://nr.harvard.edu/nr/bookc.html: Cambridge University Press, 2nd edn., 1992.

Price, G. R. "Selection and Covariance." *Nature*, vol. 227, pp. 520–521, 1970.

Punch, W. F., Goodman, E. D., Pei, Min, Chia-Shun, Lai, Hovland, P., and Enbody, R. "Further Research on Feature Selection and Classification using Genetic Algorithms." In *Proceedings of the International Conference on Genetic Algorithms*, pp. 557–564. 1993.

Quinlan, J. R. "Inferring Decision Trees Using the Minimum Description Length Principle." *Information and Computation*, vol. 80, pp. 227–248, 1989.

Quinlan, J. R. *C4.5: Programs for Machine Learning*. Morgan Kauffman, 1993.

Quinlan, J. R. "Improved Use of Continuous Attributes in C4.5." *Journal of Artificial Intelligence Research*, vol. 4, pp. 77–90, 1996.

Quinlan, J. Ross and Cameron-Jones, R. Mike. "Induction of Logic Programs: FOIL and Related Systems." *New Generation Computing*, vol. 13(3&4), pp. 287–312, 1995.

Rasmussen, Carl Edward, Neal, Radford M., Hinton, Geoffrey, van Camp, Drew, Revow, Michael, Ghahramani, Zoubin, and andRob Tibshirani, Rafal Kustra. "DELVE: Data for Evaluating Learning in Valid Experiments." 1996. Data sets, learning environment and learning algorithms available at http://www.cs.utoronto.ca/~delve/.

Riedmiller, Martin and Braun, Heinrich. "A Direct Adaptive Method for Faster Backpropagation Learning: the RPROP Algorithm." In *1993 International Conference on Neural Networks*, vol. 1, pp. 586–591. IEEE, 1993.

Ripley, B. D. "Neural Networks and Related Methods for Classification." Tech. rep., Department of Statistics, University of Oxford, 1992.

Ripley, B. D. "Flexible Non-linear Approaches to Classification." In V. Cherkassky, J. H. Friedman, and H. Wechsler, eds., *From Statistics to Neural Networks*, pp. 105–126. Springer-Verlag, 1994.

Ripley, B. D. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

Rogova, Galina. "Combining the Results of Several Neural Network Classifiers." *Neural Networks*, vol. 7(5), pp. 777–781, 1994.

Ronald, Simon, Asenstorfer, John, and Vincent, Millist. "Representational Redundancy in Evolutionary Algorithms." In *Proceedings of the IEEE Conference on Evolutionary Computation*, pp. 631–636. 1995. I went to this one too.

Rosca, Justinian P. "An Analysis of Hierarchical Genetic Programming." Technical Report 566, University of Rochester, Rochester, NY, USA, 1995*a*.

Rosca, Justinian P. "Genetic Programming Exploratory Power and the Discovery of Functions." In John Robert McDonnell, Robert G. Reynolds, and David B. Fogel, eds., *Evolutionary Programming IV Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pp. 719–736. San Diego, CA, USA: MIT Press, 1995*b*.

Rosca, Justinian P. and Ballard, Dana H. "Genetic Programming with Adaptive Representations." Tech. Rep. TR 489, University of Rochester, Computer Science Department, Rochester, NY, USA, 1994.

Ross, T.J. *Fuzzy Logic with Engineering Applications.* McGraw-Hill, Inc., 1995.

Ryan, Conor. "Pygmies and Civil Servants." In Kenneth E. Kinnear, Jr., ed., *Advances in Genetic Programming*, chap. 11, pp. 243–263. MIT Press, 1994.

Schaal, Stefan and Atkeson, Christopher C. "From Isolation to Cooperation: An Alternative View of a System of Experts.", 1995. Submitted to *Neural Information Processing Systems 1995.*

Schaffer, Cullen. "A Conservation Law for Generalization Performance." In William W. Cohen and Haym Hirsh, eds., *Machine Learning: Proceedings of the Eleventh International Conference*, pp. 259–265. New Brunswick, N.J.: Rutgers University, 1994.

Scott, M. J. J., Niranjan, M., and Prager, R. W. "Parcel: Feature Subset Selection in Variable Cost Domains." Tech. Rep. 323, Cambridge University Engineering Department, 1998.

Sebag, Michele, Maitournam, Habibou, and Schoenauer, Marc. "Identification of Mechanical Behaviour by Genetic Programming Part I: Rheological Formulation." Tech. rep., Ecole Polytechnique, 91128 Palaiseau, France, 1995.

Sethi, Ishwar K. "Neural Implementation of Tree Classifiers." *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25(8), pp. 1243–1249, 1995.

Sharman, Ken C. and Esparcia-Alcazar, Anna I. "Genetic Evolution of Symbolic Signal Models." In *Proceedings of the Second International Conference on Natural Algorithms in Signal Processing, NASP'93.* Essex University, 1993.

Sharman, Ken C., Esparcia Alcazar, Anna I., and Li, Yun. "Evolving Signal Processing Algorithms by Genetic Programming." In A. M. S. Zalzala, ed., *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, vol. 414, pp. 473–480. Sheffield, UK: IEE, 1995.

Sherrah, Jamie. "Automatic Feature Extraction using Genetic Programming." In John R. Koza, ed., *Late Breaking Papers at the 1997 Genetic Programming Conference*, p. 298. Stanford University, CA, USA: Stanford Bookstore, 1997.

Sherrah, Jamie and Jain, Ravi. "Classification of Heart Disease Data using the Evolutionary Pre-Processor." In *Engineering Mathematics and Applications Conference.* University of Adelaide, 1998. (accepted for publication).

Sherrah, Jamie R., Bogner, Robert E., and Bouzerdoum, Abdesselam. "Automatic Selection of Features for Classification using Genetic Programming." In V. L. Narasimhan and L. C. Jain, eds., *Proceedings of the Australian New Zealand Conference on Intelligent Information Systems*, pp. 284–287. IEEE Press, 1996.

Sherrah, Jamie R., Bogner, Robert E., and Bouzerdoum, Abdesselam. "The Evolutionary Pre-Processor: Automatic Feature Extraction for Supervised Classification using Genetic Programming." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 304–312. Stanford University, CA, USA: Morgan Kaufmann, 1997.

Sinkkonen, Janne. "What is the Curse of Dimensionality?" In *comp.ai.neural-nets FAQ, Part 2 of 7: Learning*. http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-8.html: The Internet FAQ Consortium, 1998.

Soule, Terence, Foster, James A., and Dickinson, John. "Code Growth in Genetic Programming." In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 215–223. Stanford University, CA, USA: MIT Press, 1996.

Spears, William. "Adapting Crossover in Evolutionary Algorithms." In *Proceedings of the Evolutionary Programming Conference*, pp. 367–384. 1995.

Stearns, Beth. "Integrating Native Code and Java Programs." In *The Java Tutorial*. http://java.sun.com/docs/books/tutorial/: Sun Microsystems, 1997.

Syswerda, Gilbert. "Uniform Crossover in Genetic Algorithms." In J. D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 2–9. San Mateo, California USA: Morgan Kaufmann, 1989.

Tackett, Walter Alden. "Genetic Programming for Feature Discovery and Image Discrimination." In Stephanie Forrest, ed., *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pp. 303–309. University of Illinois at Urbana-Champaign: Morgan Kaufmann, 1993.

Tackett, Walter Alden. *Recombination, Selection, and the Genetic Construction of Computer Programs*. Ph.D. thesis, University of Southern California, Department of Electrical Engineering Systems, 1994.

Telfar, Grant. *Generally Applicable Heuristics for Global Optimisation: an Investigation of Algorithm Performance for the Euclidean Traveling Salsman Problem*. Master's thesis, Institute of Statistics and Operations Research, Victoria University of Wellington, 1994.

Teller, Astro. "Evolving Programmers: The Co-evolution of Intelligent Recombination Operators." In Peter J. Angeline and K. E. Kinnear, Jr., eds., *Advances in Genetic Programming 2*, chap. 3, pp. 45–68. Cambridge, MA, USA: MIT Press, 1996.

Teller, Astro and Andre, David. "Automatically Choosing the Number of Fitness Cases: The Rational Allocation of Trials." In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 321–328. Stanford University, CA, USA: Morgan Kaufmann, 1997.

Teller, Astro and Veloso, Manuela. "Algorithm Evolution for Face Recognition: What Makes a Picture Difficult." In *International Conference on Evolutionary Computation*, pp. 608–613. Perth, Australia: IEEE Press, 1995*a*.

Teller, Astro and Veloso, Manuela. "Program Evolution for Data Mining." *The International Journal of Expert Systems*, vol. 8(3), pp. 216–236, 1995*b*.

Teller, Astro and Veloso, Manuela. "PADO: A New Learning Architecture for Object Recognition." In Katsushi Ikeuchi and Manuela Veloso, eds., *Symbolic Visual Learning*, pp. 81–116. Oxford University Press, 1996.

Tham, C K. "On-Line Learning Using Hierarchical Mixtures Of Experts." *Proceedings of the IEE Conference on Artificial Neural Networks*, pp. 347–351, 1995.

Thrun, S.B., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., Jong, K. De, Dzeroski, S., Fahlman, S.E., Fisher, D., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R.S., Mitchell, T., Pachowicz, P., Vafaie, Y. Reich H., de Welde, W. Van, Wenzel, W., Wnek, J., , and Zhang, J. "The MONK's Problems - A Performance Comparison of Different Learning algorithms." Tech. Rep. CS-CMU-91-197, Carnegie Mellon University, 1991.

Tibshirani, R. "Bias, Variance and Prediction Error for Classification Rules." Tech. rep., Department of Preventive Medicine and Biostatistics and Department of Statistics, University of Toronto, 1996.

Vafaie, Haleh and De Jong, Kenneth A. "Robust Feature Selection Algorithms." In *Proceedings of the International Conference on Tools with AI*, pp. 356–364. Boston, MA: IEEE Computer Society Press, 1993.

Vafaie, Haleh and Imam, I. "Feature Selection Methods: Genetic Algorithms vs. Greedy-like Search." In *Proceedings of the International Conference on Fuzzy and Intelligent Control Systems.* Louisville, KY, 1994.

Vavak, F. and Fogarty, T. C. "A Comparative Study of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments." In *Proceedings of the Society for the Study of Artificial Intelligence & Simulation of Behaviour Workshop on Evolutionary Computing*, pp. 301–307. University of Sussex, 1996.

Waggoner, Ben. "Jean-Baptiste Lamarck (1744-1829)." http://www.ucmp.berkeley.edu/history/-lamarck.html, 1996.

Wall, Matthew B. "GAlib: A C++ Genetic Algorithm Library (ver. 2.4.2)." http://lancet.mit.edu/ga/, 1996. Copyright 1994-5 Massachusetts Institute of Technology.

Wampler, Bruce. "V - A Freeware Portable C++ GUI Framework for Windows, X, and OS/2." http://www.objectcentral.com/, 1998.

Webb, Andrew R and Lowe, David. "The Optimised Internal Representation of Nultilayer Classifier Networks Performs Nonlinear Discriminant Analysis." *Neural Networks*, vol. 3, pp. 367–375, 1990.

Weisstein, Eric. *The CRC Concise Encyclopedia of Mathematics.* Online at http://www.astro.virginia.edu/-~eww6n/math/math0.html: CRC Press, 1998.

Whigham, P. A. "A Schema Theorem for Context-Free Grammars." In *1995 IEEE Conference on Evolutionary Computation*, vol. 1, pp. 178–181. Perth, Australia: IEEE Press, 1995.

White, Matt and Fahlman, Scott E. "Carnegie Mellon University Neural Network Benchmark Database." ftp://ftp.cs.cmu.edu/afs/cs/project/connect/bench, 1993.

Widrow, Bernard and Lehr, Michael A. "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation." *Proceedings of the IEEE*, vol. 78(9), pp. 1415–1442, 1990.

Winston, P.H. *Artificial Intelligence.* Addison-Wesley, 3rd edn., 1992.

Wolpert, David H. "On Bias Plus Variance." SFI TR 95-08-074, Santa Fe Institute, 1995.

Wolpert, David H. "The Lack of *A Priori* Distinctions Between Learning Algorithms.", 1996. Available via ftp://ftp.santafe.edu/pub/dhw_ftp/nfl.1.ps.Z.

Wolpert, David H. and Macready, William G. "No Free Lunch Theorems for Search." Tech. rep., The Santa Fe Institute, 1996.

Wolpert, David H. and Macready, William G. "No Free Lunch Theorems for Optimization." *IEEE Transactions on Evolutionary Computation*, vol. 1(1), pp. 67–82, 1997.

Wu, Annie S. and Lindsay, Robert K. "A Survey of Intron Research in Genetics." In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, eds., *Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation*, vol. 1141 of *LNCS*, pp. 101–110. Berlin, Germany: Springer-Verlag, 1996.

Young and Fu. *Handbook of Pattern Recognition and Image Processing.* Orlando:Academic Press, 1986.

Zhang, Byoung-Tak. "Acclelerated Learning by Active Example Selection." *International Journal of Neural Systems*, vol. 5(1), pp. 67–75, 1994.

Zhang, Byoung-Tak and Mühlenbein, Heinz. "Genetic Programming of Minimal Neural Nets Using Occam's Razor." In Stephanie Forrest, ed., *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pp. 342–349. University of Illinois at Urbana-Champaign: Morgan Kaufmann, 1993.

Zhang, Byoung-Tak and Mühlenbein, Heinz. "Balancing Accuracy and Parsimony in Genetic Programming." *Evolutionary Computation*, vol. 3(1), pp. 17–38, 1995.

Zhang, Byoung-Tak, Ohm, Peter, and Mühlenbein, Heinz. "Learning to Predict by Evolutionary Neural Trees." *Proceedings of the World Congress on Neural Networks*, vol. 1, pp. 823–826, 1995.

Zheng, Zijian. "A Benchmark for Classifier Learning." Tech. rep., Basser Department of Computer Science, The University of Sydney, 1993.