OXFORD CHEMISTRY MASTERS

Series Editors

RICHARD G. COMPTON
University of Oxford

STEPHEN G. DAVIES
University of Oxford

JOHN EVANS
University of Southampton

OXFORD CHEMISTRY MASTERS

1. A. Rodger and B. Nordén: *Circular dichroism and linear dichroism*
2. N.K. Terrett: *Combinatorial chemistry*
3. H.M.I. Osborn and T.H. Khan: *Oligosaccharides: Their chemistry and biological roles*
4. H. Cartwright: *Intelligent data analysis in science*

Intelligent Data Analysis in Science

Edited by
HUGH CARTWRIGHT
University of Oxford

OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Great Clarendon Street, Oxford OX2 6DP
Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide in

Oxford New York

Athens Auckland Bangkok Bogotá Buenos Aires Calcutta
Cape Town Chennai Dar es Salaam Delhi Florence Hong Kong Istanbul
Karachi Kuala Lumpur Madrid Melbourne Mexico City Mumbai
Nairobi Paris São Paulo Singapore Taipei Tokyo Toronto Warsaw

with associated companies in
Berlin Ibadan

Oxford is a registered trade mark of Oxford University Press
in the UK and in certain other countries

Published in the United States
by Oxford University Press, Inc., New York

First published 2000

A catalogue record for this title is available from the British Library

Library of Congress Cataloging in Publication Data
(Data available)

1 3 5 7 9 10 8 6 4 2

ISBN 0 19 850233 8

Typeset by EXPO Holdings, Malaysia

Printed in Great Britain on acid free paper by
The Bath Press, Avon

Series Editors' Foreword

Oxford Chemistry Masters are designed to provide clear and concise accounts of important topics – both established and emergent – that may be encountered by chemistry students as they progress from the senior undergraduate stage through postgraduate study to leadership in research. These Masters assume little prior knowledge, other than the foundations provided by an undergraduate degree in chemistry, and lead the reader through to an appreciation of the state-of-the-art in the topic whilst providing an entreé to the original literature in the field.

In this volume Hugh Cartwright has brought together an authoritative group of contributors to provide a clear and comprehensive account of the use of artificial intelligence for *Intelligent Data Analysis in Science*. This book will interest novices, initiates and masters in the field who wish to exploit the new and powerful techniques described and to use them in a diversity of applications within chemistry and beyond.

Richard G. Compton
Stephen G. Davies
John Evans

Preface

That computers have revolutionized scientific research is beyond doubt. Calculations can now be carried out at speeds that would have been unimaginable a few decades ago. However, just as crucially, computers have opened up a new world of analysis through the development of intelligent methods.

Computer programs employing Artificial Intelligence (AI) are enjoying a period of explosive growth. AI programs can now be found embedded in scientific instruments, taking control of industrial production lines, and built into a wide variety of commercially available software. AI constitutes a set of tools of great versatility and power, and this is well illustrated by the contributions to this text which brings together some of the leading researchers in this emerging field.

Each chapter includes theory, practical details of implementation, and the application of AI to the solution of real scientific problems. The extensive lists of references at the end of each chapter will be indispensable to all those who are encouraged to explore the subject further.

The potential of AI in science is immense. The methods and applications discussed in this text will consequently be of wide interest to those both in the physical and natural sciences and in mainstream AI research.

Oxford H.C.
January 2000

Contents

Contributors

Samir Agrawal
School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752, USA.

David Andre
Computer Science Division, University of California, Berkeley, California, USA.

Forrest H. Bennett III
Chief Scientist, Genetic Programming Inc., Los Altos, California 94023, USA.

Carlos A. Coello Coello
Engineering Design Centre, University of Plymouth, Drake Circus, Plymouth, Devon PL4 8AA, England.

Royston Goodacre
Institute of Biological Sciences, University of Wales, Aberystwyth, Ceredigion, SY23 3DD, Wales.

D. Brynn Hibbert
Department of Analytical Chemistry, The University of New South Wales, P. O. Box 1, Kensington, New South Wales, Australia 2033.

Erik Johnson
School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752, USA.

Hillol Kargupta
School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752, USA.

Martin A. Keane
Chief Scientist, Econometrics Inc., 111 E. Wacker Dr., Chicago, Illinois 60601, USA.

John R. Koza
Section of Medical Informatics, School of Medicine, Stanford University, Stanford, California, 94305, USA.

Sharbari Lahiri
Department of Chemistry, University of Western Ontario, London, Ontario, Canada N6A 5B7

Fuji Lai
Department of Electrical Engineering, Duke University, Box 90291, Durham, North Carolina 27708-0291, USA.

Mary Mulholland
University of Technology, Sydney, Australia.

Ian C. Parmee
Engineering Design Centre, University of Plymouth, Drake Circus, Plymouth, Devon PL4 8AA, England.

Eleonora Riva Sanseverino
School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752, USA.

Martin J. Stillman
Department of Chemistry, University of Western Ontario, London, Ontario, Canada N6A 5B7.

Paul P. Wang
Department of Electrical Engineering, Duke University, Box 90291, Durham, North Carolina 27708-0291, USA.

Andrew H. Watson
Engineering Design Centre, University of Plymouth, Drake Circus, Plymouth, Devon PL4 8AA, England.

1 Introduction to intelligent data analysis

D. Brynn Hibbert

1. Introduction

I am an Analytical Chemist, and it comes as a surprise to my new students that hardly anyone in the world wants to know the concentration of X (where X is a compound of their choice) *per se*. People want to know the answers to questions such as 'Can I drink the water?', 'Is it safe to live next door to a nuclear power station?', or 'Should I invest in the proposed new gold mine?'. The answer to each of these sample questions requires chemical analysis, allied with inputs from other science-based disciplines, but the ultimate answer is not in parts per million, but a simple 'yes' or 'no'. Intelligent data analysis, therefore, has as an ultimate goal to take inputs from scientific endeavor (signals from instruments), with other relevant information, and to turn these inputs into reasoned answers. It is the purpose of this book to introduce the reader to the mainstream methods of extracting information from data, and the purpose of this introductory chapter to set the scene, define the terms, and possibly to indulge in some crystal ball gazing.

Science at the start of the twenty first century goes about its business via a plethora of instruments that interact with the outside world. Direct observation has been superseded by mediated methods that amplify, change, and pick out aspects of the world. The optics of a simple telescope (a straightforward mediated method) may be understandable, but the complex signal from a high-field, pulsed nuclear magnetic resonance spectrometer will require considerable manipulation to extract the useful information about different nuclei in the sample. Even then there may be further steps in using that information for the benefit of society.

2. Towards autonomous, intelligent machines (AIMs)

The present interest in intelligent data analysis is but a step on the way to intelligent instruments and finally autonomous, intelligent machines. The latter, the robots of science fiction, will have generalized mission statements rather than specific directives. They will collect data, intelligently analyse them, and act on the results. Unlike the robots of science fiction they are unlikely to be humanoid, nor very large. Imagine a nano-machine injected into a patient's bloodstream with a goal to repair a faulty heart valve and then return for retrieval (or perhaps self destruct). The machine would need a powerful computer processor, sensors for compounds in the blood, temperature, and pressure. It must have some mechanical way of

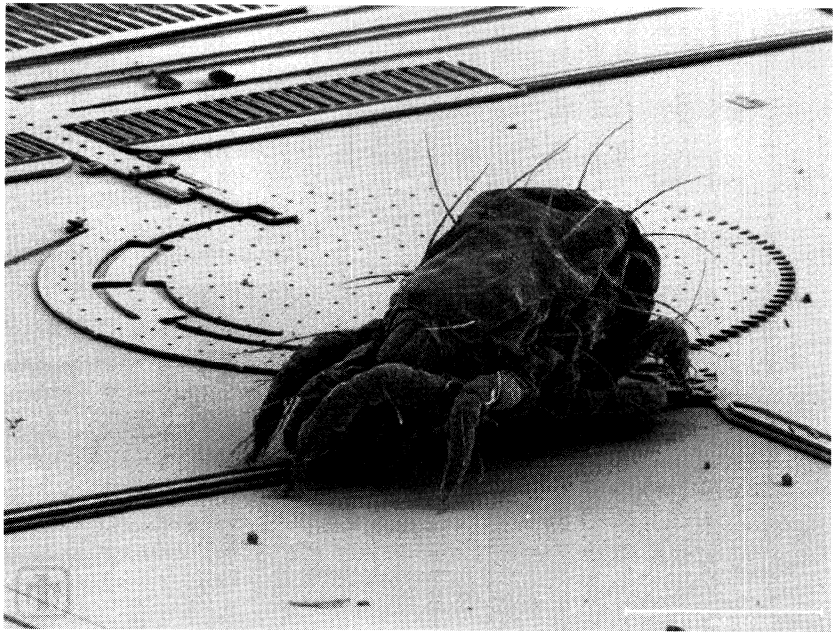

Fig. 1.1 Electron micrograph of a spider mite on a micro lock mechanism. Photograph courtesy of Sandia National Laboratories' Intelligent Micromachine Initiative, http://www.mdl.sandia.gov/Micromachine.

orienting and propelling itself, and tools for completing the task (miniature lasers, protein synthesis kit for tissue repair). Nearly all this technology exists. Much of it has been miniaturized, including gears, wheels, and other moving parts (Fig. 1.1).

Chemical sensors have been developed for many of the ions (H^+, Na^+, K^+, Ca^{2+}, Cl^-) and biological species (enzyme electrodes for glucose, amino acids, hormones, etc) likely to be encountered. Arrays of sensors are a prime target for intelligent data analysis and will be discussed at greater length below. Feedback and control has been demonstrated. A simple mobile robot has been equipped with an 'artificial nose' (an array of vapour sensors) and trained to follow trails of camphor painted on a factory floor (Deveza *et al.* 1994). The algorithm used gives a path similar to that followed by a male lobster, tracking down the source of a pheromone plume release by an egg-laden female (Atema 1995).

Possibly the greatest lacunae in the, admittedly fanciful, account given above is in the software whereby data from the sensors can be analysed and turned into actions that satisfy the mission's primary goals, i.e. intelligent data analysis.

Figure 1.2 attempts to bring together this discussion to show the relation between the different levels of intelligent instrumentation, and where intelligent data analysis is used to achieve these wider goals.

The solid line encloses an intelligent instrument. The end user looses the instrument on the world and it then takes appropriate measurements, con-

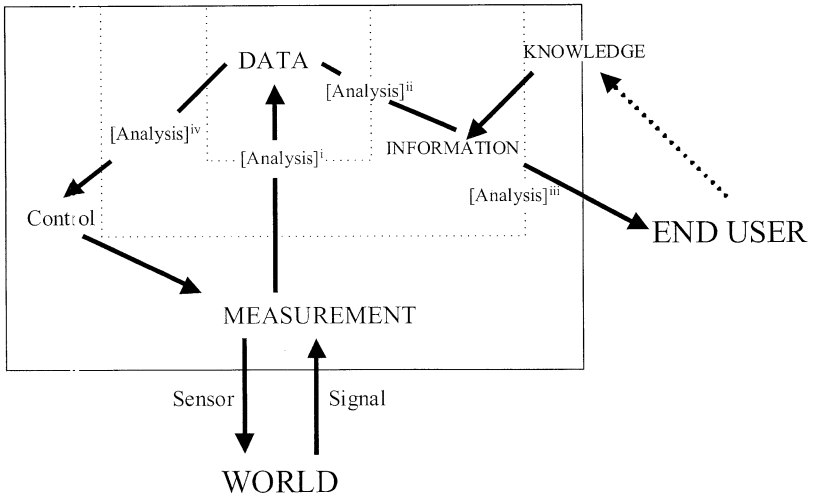

Fig. 1.2 Data, instruments, and intelligent analysis. See text for discussion.

cerns itself with validation, calibration, and so on, and returns results, advice, and performs such actions as are required. The first dotted line contains an instrument that the user controls and uses to take measurements. There is still full scope for intelligent data analysis using methods that are described in this book. The inner rectangle shows the present position of many instruments, with some analysis being performed internally but the user being required to understand and control the instrument, and also perform most of the end data analysis off line.

3. Data

Many a criminal case has revolved around data — blood tests on stains found at the scene of the crime, the level of alcohol in a driver's blood, the speed of a suspect's motor car. Because of the need to be certain of what came out of a forensic scientist's black box, lawyers have caused the notion of 'primary data' to be distinguished from 'derived data'. In the past, the primary data could have been the reading on the dial of an instrument, faithfully recorded in the scientist's laboratory notebook. Now, in the age of electronic instruments, the primary data may well be the contents of a file residing on a hard disk. Whatever form the primary data takes, it provides us with a starting point for our discussion of intelligent data analysis.

All instruments output a signal in the form of a voltage (or occasionally a current, that is readily converted to a voltage). This may cause the needle on a meter to move, it may cause a pen to trace out a line on a chart recorder, or it may be converted to a digital signal via an analogue to digital converter (ADC) and thus be relayed to a computer.

Intelligent data analysis may start its work on this raw signal, or on the product of some algorithmic manipulation ([Analysis]^{i} in Fig. 1.2). The

output of a pH meter is an example of such a conversion. It is obtained from the voltage between a glass electrode and a reference electrode via the equation

$$\mathrm{pH} = a - b\,V \tag{1}$$

a is a constant, determined by measuring the voltage (V) in a solution of known, or defined pH, and b is calculated from a theoretical treatment of the response of such electrodes (the Nernst equation (Hibbert 1993)) and the measured temperature. Although the intelligence shown by Nernst in deriving his equation in the 1890s was considerable, no one would ascribe intelligence to the operation of the internal electronics in the pH meter. Wherein, therefore, lies intelligence in data analysis?

4. Intelligence

Philosophers and computer scientists have made a tidy business of discussing the nature of artificially intelligent machines and their relation to human intelligence. The sections on knowledge-based reasoning and expert systems confront directly the question 'What is intelligence?', and other chapters in this text build on the notion implicit in the title that there exists something that could be called 'intelligent data analysis'. The much-criticised definition of artificial intelligence given by Minsky (1968) 'the science of making machines do things that would require intelligence if done by men' may, apart from the inherent sexism, serve us here. We believe that the interpretation of scientific data requires intelligence, and thus computer software that can take low-level data and spit out high-level information, advice, or simply more useful data may be described as 'intelligent'. In adopting this approach we may be accused of a 'heads-down, pure engineering' view of artificial intelligence (Whitby 1988) but it will allow us to make progress on the central issue of whether we can use computer procedures in the sciences that can substantially upgrade the information availability of data. The philosophical foundations of this book may at least be on a par with the modern concepts of the 'intelligent micromachine initiative' (see Fig. 1.1) and somewhat ahead of 'intelligent polymers'[1].

5. Heuristics and learning

The exercise of human intelligence often arises when applying general knowledge to fill in the gaps in understanding situations for which only partial information is available. The data from most experiments may, theoretically, have a number of interpretations but in the context of a given situation humans see the obvious one, often being guided by expectations about the outcome. The intervention of an intelligent human, and what distinguishes

[1]There exists at the University of Wollongong, New South Wales, Australia the 'Intelligent Polymer Research Centre'.

levels of expertise in humans, is a subtle blend of experience and understanding of the field in question. Expertise is gained by exposure to situations for which the answer is known (or becomes apparent), direct teaching, and generally trying and failing. This is the most difficult 'intelligence' to bring to data analysis.

6. Data analysis and the embodiment of intelligence

In using a term such as 'increase the information availability' of data, we incorporate the idea that, at one level, data analysis can be simply a reorganization of data. A principal components analysis (PCA) of a set of spectra may reveal groupings of objects or variables that are not apparent from an examination of the original data (Martens and Naes 1989). PCA is a linear matrix manipulation of the data. Nothing is added or removed, the data is reorganized and in doing so features are revealed allowing the intelligence of the observer to extract the necessary information ([Analysis][ii] in Fig. 1.2).

Modern trends in scientific instrumentation have been to increase the amount of data enormously. A human plotting data a point at a time read from an instrumental dial lags far behind an analogue to digital converter accessing many channels at a megahertz rate. Diode array detectors, using charge coupled device technology, capture many spectra per minute at hundreds of wavelengths. The information so obtained has increased the power of HPLC to discriminate between closely eluting compounds. The millions of numbers representing absorbances at given wavelengths and times cannot be understood when viewed as raw data. At least a three-dimensional visualization and further intelligent analysis is required to make sense of what is being seen.

Intelligent data analysis really comes into its own when the data issuing from the instrument is augmented by prior knowledge or understanding ([Analysis][iii] in Fig. 1.2). The dotted line from the End User in Fig. 1.2 shows that knowledge is required if the user is to perform her own analysis, but the instrument could already be imbued with such knowledge, input via the knowledge base of an expert system, or learned from earlier experience.

7. Traceability, calibration, and validation

Traceability to the Système Internationale (SI) is common in all fields of physical measurement but has only recently been identified as a major problem in chemistry (Kaarls and Quinn 1997). Commercial (through trade) and public (health and environment) pressures demand reliable measurements that can only be assured if those measurements are traceable to agreed international references. Physical measurements of time, length, and weight have long been associated with international standards. Electrical potential and current are also traceable. More recently traceability of chemical measurements has been extensively discussed, both in terms of possible redefinition of the gram in terms of the SI measure of the amount of substance (the mole)

and practical realization of this by establishment of certified reference laboratories (de Bièvre and Taylor 1997).

The reliance on apparatus that does not directly observe Nature, implies the need for calibration, namely the procedure by which the relationship between the instrumental response and the level of the measurand is established. If an instrument returns the concentration of a component, no matter in how complex a manner, traditional calibration could be used with certified reference materials providing the traceability to international standards. If, however, the intelligence is taken one step further to offer advice based on the direct results of the measurements, then it is hard to see how metrologically valid calibration could be effected. An example would be the establishment of the authenticity of olive oil, based on the output of an array of vapour sensors and a neural network trained on a number of genuine and ersatz samples. No single compound is explicitly determined, rather it is the ensemble of vapours that triggers the real/fake result. It may be argued that because no actual concentration is measured then there is no metrology problem to consider. However if instruments in the future are to be seen as increasingly intelligent, then some means of assessing the calibration must be developed. Sobolev and Aumala (1996) have addressed the problem of metrological support in intelligent systems, defining their concept of Metrological Automatic Support (MAS). While concentrating on intelligent instruments for which measurements are without ambiguity (by implication physical rather than chemical measurements), they identify complexity and the ability of intelligent systems to reconfigure themselves as reasons why traditional metrological approaches may fail.

That an intelligent instrument may arrive at an answer from a number of pathways through the possible configurations of the instrument and the near impossibility of establishing the metrological characteristics of those possible structures has led to the concept of internal or autonomous verification. This has the advantage of tracking changes in the system itself and thus maintaining a chain of calibration even if the instrument changes the structure of the measurement chain. Models of the working of sensors and other parts of an intelligent instrument are required, and these may be set up and tested via virtual instruments (see below).

There is a clear challenge to the writers of intelligent software to establish the validity of their code. Validation is the demonstration that a process is acceptable for its intended purpose (Green 1996). The following, with an emphasis on the analysis of pharmaceutical compounds, are identified as being generally required for validation of an analytical method: specificity, accuracy, precision, detection limit, quantitation limit, linearity, and robustness. If the output of intelligent data analysis is a concentration then most of the validation criteria could be used. In the more complex world of intelligent machines linearity of calibration would be replaced by a suitable statistical measure of the goodness of the regression, perhaps established by cross validation. Interlaboratory trials could establish measures of precision even if an uncertainty audit, with propagation of uncertainty of the steps in the analysis through to the final result, is not possible.

Accuracy is usually measured in one of four ways. The most metrologically sound method is to compare the analysis with that of a certified reference material (CRM) of near the same concentration and composition. For many samples such a well-characterized CRM is not available. Second, the results may be compared with the analysis using another validated test method. This also encompasses the idea of using a certified reference laboratory to independently analyse a sample. However exactly equivalent methods are not always available. Thirdly, a known amount of the analyte (a spike) may be added to a blank matrix and the recovery determined. This is common in the pharmaceutical industry. If the matrix cannot be made up independently to give a blank to which analyte may be added, the final method is that of standard addition to the sample, followed by a calculation of the recovery.

The measure of robustness is fraught with danger for methods based on intelligent data analysis. The robustness of a method is its ability to remain unaffected by small changes in the system as may be found in normal practice. Parameters such as temperature, pH, the way that the instrument is operated, and the nature of the matrix will show a natural variation, and unless they are explicitly compensated for (such as temperature) it must be established how they will affect the analytical result. It is rarely good enough to change one identified factor at a time. Any correlation between variables will lead to erroneous results. Multivariate methods that allow the analysis of target analytes in the presence of multicomponent matrices means that to be tested thoroughly, a large experimental design is probably required. As with the narrower problem of traceability discussed above, the more intelligent the system, the wider the range of situations in which it can be used, and the greater the task of validation.

In addition to the analytical validation that is required, a complex program must be validated internally, i.e. does the software deliver the output that the programmer thought it should? This is a major concern of computer science and a steady stream of much publicized computer-aided disasters does not appear to aid public confidence in the ability of software to deliver the desired results (Waldrop 1987).

8. Requirements for an intelligent sensor

For data analysis to be intelligent, the data supplied must be of a quality to allow such analysis to function properly. A number of papers in a volume of the journal *Measurement and control* were devoted to the topic of intelligent instruments (Brignell 1996). From an engineering aspect there are several qualities of an intelligent instrument that are worthy of note. Riviere *et al.* (1996) listed the properties and functionality of an intelligent sensor. These are compiled, with considerable amendment for an intelligent chemical sensor in Table 1.1.

In the first column we see that collection of data is only one function that is required of a suitably intelligent instrument. The data must be stored (in the case of large data sets this may require compression) and properly indexed. When interrogated the data must be made available, after some validation

Table 1.1 Intelligent chemical sensors. Requirements and functionality

Data manipulation to be made	Measurements	Data analysis
• Collection of data (measurement) • Storage • Distribution to authorized personnel • Deletion of data (clearing of memory)	• Measurand(s) • Physical (temperature pressure) • Internal diagnostic • Time	• Calibration • Calculation of required information and inferences (concentrations leading to advice) • Validation /diagnostics, Time/date stamp • Advice — answering wider questions with contextual understanding

that the system or person accessing the instrument has authority to do so. The memory will be freed when the data is no longer needed by the instrument, or when higher priority data must be stored.

In many present-day sensors the instrument has an obvious primary measuring device, plus other sensors that are required to compensate the measurement and internal devices to check the general integrity of the device ([Analysis][iv] in Fig. 1.2). However increasingly there is what may be called a 'total measurement process' in which a number of variables are measured simultaneously and processed to yield both information about the analyte and other diagnostic information. Array sensors used for vapours in a number of 'electronic nose' instruments are a good example of this. Thus at the level of the sensor itself, the distinction between measurand and other species has become blurred, leaving the data analysis to sort out the different output requirements.

The data analysis therefore brings together measurements on standards (calibration) and the system to provide a first level of information, which may be concentrations of the target analytes. For a complex array of measurements the software to accomplish this could well be classed as intelligent. Further intelligence is shown if the measurements are validated, by checking compliance information, making measurements on check standards, determining that measurements are within acceptable bounds, that calibration linearity is maintained, and that the basic electronics are functioning as specified. Data should have the appropriate level of auxiliary information, the minimum being a time/date stamp. Finally an ensemble of data may be analysed to provide the top level of information, such as advice to the user that may come from an expert system.

The foregoing discussion shows the serious approach taken by the engineering community to the problems created by the complexity of intelligent instruments. A neural network that uses sensory input data to determine the origin of olive oil, is a long way from a finished, validated instrument that can work in an olive oil bottling plant. Two interesting approaches to the design of intelligent instruments have recently been published. One tests and designs the instrument in a computer program, thus creating a virtual instrument that can quickly be assessed, changed, or reprogrammed before too much expensive engineering takes place. Taner and White (1996) describe a technique using virtual instrumentation for testing and analyzing an electronic sub-system.

Luttenbacher *et al.* (1996) consider the requirements of an intelligent sensor having to connect to other sensors and instruments as part of the make up of intelligent machines. To integrate an intelligent sensor into an automated system, there must exist a model of the sensor in terms of the services it can offer, the inputs required, and its general behaviour. They describe an object oriented approach to build models, and give an example using OMT (object modelling technologies).

9. Appropriate intelligent methods

Reading the chapters of this book may leave a neophyte data-analyser with the question 'which, of such powerful methods should I use?'. There are hard-line proponents of any method who maintain that all knowledge may be extracted using their technique, but there is little evidence that any one method is superior over a wide range of problems. Typically the literature abounds with papers that compare method X with method Y for a restricted problem, and for which X is obviously better suited than Y anyway. The choice of data analysis method must rest on two considerations, the nature of the data and the nature of the required output. Practically there is a third factor, namely the availability and user friendliness of the software, which tends to hinder the widespread use of new methods.

Figure 1.3 attempts to show the relationship between the three main areas of data analysis, statistical multivariate analysis, neural networks and logic-based programming, and the nature of the data and problem.

Data may be numerical, either continuous or discrete[2], or symbolic. Logical manipulation of symbols (words, sentences, etc), and the output of advice is usually thought of as the province of expert systems, knowledge-based reasoning, and the like. At the other extreme, the treatment of numerical data with a quantitative output is often treated by statistically based methods such as principal components analysis, and regression, partial least squares regression, or ridge regression. Classification based on numerical data may be accomplished by methods such as discriminant analysis,

[2]Because of the use of analog to digital converters all data from instruments is discrete, but if the discretization is fine enough, methods are not disadvantaged by treating the data as continuous

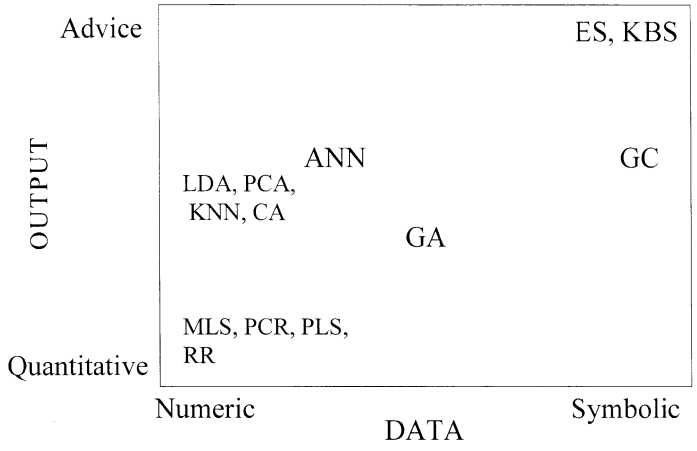

Fig. 1.3 Intelligent data analysis methods arranged in terms of the nature of the data and the required output. Key to methods: LDA, linear discriminant analysis; KNN, K-nearest neighbours; PCA, principal components analysis; CA, cluster analysis; MLS multivariate linear regression; PCR, principal components regression; PLS, partial least squares; RR, ridge regression; ANN, artificial neural network; GA, genetic algorithm; GC, genetic classifier; KBS knowledge based system; ES, expert system.

SIMCA, K-nearest neighbour, or cluster analysis. Artificial neural network methods take up much of the middle ground. They are numerically based but are best suited to classification problems. The inherent non-linearity of neural networks and their flexibility has led to their wide use in data analysis. Evolutionary optimizers such as genetic algorithms are used to discover optimum parameters of the system and also show great flexibility through different codings of the problem. They may also be used to optimize another data analysis method, and thus may be considered as providing meta-intelligent data analysis. Methods can be used effectively in combination. Induction methods based on information theory can be used to provide the rules for an expert system, and neural networks have been the classification front end to expert systems in speech recognition. It should be noted, however, that with sufficient determination any method can be used for any data. Thus a purely numerical regression technique such as partial least squares that is most comfortable in determining continuous numerical variables can be used to, for example, classify diseases from blood analysis data. Similarly by binning ranges of an output variable a classifier may be made to yield a number.

10. Configuration of ion chromatography — a case study in the comparison of methods

An example of the comparison of many different analysis methods for a single problem comes with a series of papers on the configuration of an instrument to separate a mixture of ions by a group at the University of New

South Wales in Sydney, Australia (Mulholland *et al.* 1993, 1995*a*, 1995*b*, 1996; van Kampen 1997; Ramadan 1998*a*, 1998*b*). The design of a suitable ion chromatography (IC) method requires knowledge of the chemistry of ions and their interaction with column materials in different carrier solutions and the possibilities for their detection. Different chromatographic mechanisms can be applied to separate ions and each of these requires a considered choice of method conditions. By its nature IC is used to analyse mixtures and so there is infinite variety in the possible solutions that will be presented to the method. In addition the application requiring the chemical analysis may impose its own constraints, for example environmental samples may have low concentrations of the analyte ions and the amount of biological samples available may be small. Previous applications have applied expert system technology to solve problems in chromatography but IC had not been tackled until the comprehensive database produced by Haddad and Jackson (1990) provided an ideal source of data on this subject. A set of nineteen attributes were defined to characterize a system. Each attribute has a number of possible values. The attributes give information both about the sample (number of solutes, uv-adsorbing, solubility, application, presence of sulfate ...) and about the chromatographic method (detector, mobile phase, pH, mechanism of the separation ...). For each attribute it may not be clear what may be known by the user and what is required to be supplied by the program. For example, although the detector may normally be an attribute to be predicted, if a laboratory only possesses one type of detector, then that must be a given attribute and the method should, if possible, be determined around it.

The database consisted of previous published IC methods covering the literature up to 1990. The total number of cases was 14 103. The database was not amended in any way. Thus it contained errors, duplications, and conflicting advice (more than one method may be proposed for a particular case). The database also spanned ten years, during which time the subject advanced considerably, so that a method suggested in 1980 may have been superseded by 1990. Finally, it must be noted that a stunningly good method will probably only be published once. What tends to appear in the literature are niche methods and ones with novel, but not necessarily useful, methodology. For these reasons, therefore, this represented an excellent 'real world' test of the method of data analysis. The project commenced as a suitable problem for testing the use of an expert system methodology called 'ripple down rules'. The extent of the database also meant that building the system by hand would be too time consuming, and so initially two induction methods, C4.5 and INDUCT provided an automatic way to create the expert system (Mulholland *et al.* 1995, 1996). A self-organizing neural network and a genetic algorithm classifier were then applied to the database (Mulholland *et al.* 1995*b*, 1996). To address the question of whether linear statistical methods could be used on the problem, principal components regression and linear discriminant analysis were applied to a suitably coded database (Ramadan *et al.* 1998). The problem of using statistical methods requiring continuous variables for such a classification problem is well known (Sharaf *et al.* 1986). It may be argued that these methods are simply not suited to discrete data, but as principal components analysis and discriminant analysis

have become so widespread in their use, if they could be shown to work it would bring a range of problems within the purview of mainstream chemometrics.

There are two coding options. One is to assign a cardinal to each possible value of the attribute. Thus for the class of ion, 1 could code for organic ions and 2 for inorganic ions. Unfortunately these labels bear no relationship to each other. However the charge on an ion conveniently fits this coding, taking values from –4 to +4. The required output may be similarly coded, for example if the detector was being classified then an output of 1 would be the choice of a conductivity detector, 2, a UV detector and so on. Because the classification methods give a decimal value, the output needs to be rounded to the nearest cardinal. This shows the difficulty of using this coding, as an output of 1.5 does not really mean 'choose between detector number 1.0 and detector number 2.0'. A better way of coding the input and output is to assign a variable taking a value between 0 and 1 for every possible value of each attribute. Ideally for each attribute the output registers 1 for the correct choice and 0 for all others. Fractions now do have a meaning, reflecting the certainty of a given value. This coding increases the number of variables considerably (in the case of the IC problem from 19 to 118) but with a large enough train-

Table 1.2 Classification of detectors for ion chromatography by different methods

Algorithm	Percentage classified correctly	Reference
Genetic algorithm	82	van Kampen *et al.* 1996
C4.5	70	Mulholland *et al.* 1995*a*
Linear discriminant analysis with prior probability = fraction in database	69	Ramadan *et al.* 1998
Neural network	68	Mulholland *et al.* 1995*b*
INDUCT	68	Mulholland *et al.* 1995*b*
Principal components regression with binary data	60	Ramadan *et al.* 1998
Linear discriminant analysis with prior probability = 1/18	33	Ramadan *et al.* 1998
Principal components regression with cardinal data	28	Ramadan *et al.* 1998

ing set it is not prohibitive. It is not surprising that with this coding the classification of detectors was twice as successful as with cardinals (see Table 1.2) (Ramadan *et al.* 1998).

In linear discriminant analysis the discriminant score is calculated for each case for each class from which the probability that a case belongs to the given class is calculated. The discriminant function is optimized to give the maximum number of correct classifications of the training set. Once trained the discriminant function is applied to data from unknown cases and the classes duly predicted. The probability that a case with given discriminant score belongs to a particular group requires an estimate of the prior probability of the group which can be taken as proportional to the incidence of that detector in the database, or equal across the groups. In the first instance, conductivity detectors are weighted highly (7775 cases out of 12 693 use a conductivity detector) and a post-column reactor incorporating a detector (9 cases) is greatly weighted against. To classify detectors correctly that appear infrequently in the database, an equal prior probability has to be chosen. Again this shows how awkward using numerical methods is for what is essentially a symbolic problem. In an expert system little-used detectors would be catered for by unique rules that fired only occasionally. Table 1.2 compares the overall success rates of these disparate methods in choosing a detector. Apart from the methods that were not expected to work (cardinal coding in PCR and equal prior probability in LDA) there is a remarkable conformity of results, with the genetic classifier giving the best result. Inspection of the database led van Kampen *et al.* (1997) to conclude that because of the factors discussed above an 80 per cent success rate was probably the best any classification method could achieve.

11. The electronic nose — a case study in intelligent instruments

This book is organized around particular techniques that may be described as intelligent data analysis. Here we shall look at a particular application and describe the techniques that have been brought to bear on the problem. No attempt will be made to give detail of the data analysis, these topics are covered later, but it may be useful to understand the interplay between the problem, the instrumental approach, and the data analysis.

Research into the 'artificial' or 'electronic' or 'bionic' nose, is an expanding field with sensor design based on different chemical principles, a range of applications, and increasing numbers of instruments on the market (Gardner and Bartlett 1994, Hodgins and Simmonds 1995). An array of sensors, each of which responds to a number of different compounds but with different sensitivity, has the potential to differentiate among a very large number of compounds. It can be appreciated that if a sensor may yield a signal at x levels, and N sensors are grouped in an array, there are x^N possible combinations of response. For even modest values of x and N the theoretical number of combinations quickly exceeds the number of known compounds; 10^7 small organic molecules and 10^{13} polymers and proteins could be accommodated by 10 sensors returning 32 levels (5 bits). Even allowing for uncertainty in

the sensor readings leads to a calculation of 5×10^{11} compounds with 10 sensors having 10 per cent uncertainty in the response (Muller 1991). Real arrays of sensors can be highly correlated, thus reducing the information, but arrays of sensors have evident advantages over single compound sensors. The mammalian olfactory system appears to have evolved along these lines. It is not thought that one receptor in the nose codes for one type of vapour molecule. In fact it is hard to see how this could ever be selectable in an evolutionary sense as molecules new to an individual would not be perceived (i.e. there would be no mechanism for 'A smells like B' if receptors were strictly unique). At present we believe that about 1000, but possibly as many as 10 000, different receptors code for about 10 000 different smells (Bell 1996).

A generalized sensor therefore may react to generic properties of molecules such as ability to undergo oxidation, weight, size, or solubility. These properties have formed the basis of array sensors. For example the conduction of semiconductors such as tin oxide (Chiba 1990) depend on the oxidation of the impinging molecule by surface oxygen. Conducting polymers such as polypyrrole (Hierlemann *et al.* 1995) interact with organic vapours leading to changes in resistance. More sensitive are sensors that monitor mass changes of surface polymer layers as vapours partition into them. Examples of these are the quartz crystal microbalance (Barko *et al.* 1995), and surface acoustic wave devices (Hivert *et al.* 1994). Optical sensors incorporating polarity-sensitive fluorescent molecules in a polymer matrix (Dickinson *et al.* 1996) rely on the dissolution of the vapour molecule causing changes in polarity. The non-specific array is then manufactured by coating a bundle of optical fibres with films having different polymer composition and constant fluorophore. A sensor may thus be made quickly with a wide range of coatings having slightly different responses.

Whatever the transduction, the output of the array of sensors is a number of time-varying voltages. These must be calibrated to whatever problem is at hand. The interest in electronic noses tends to arise from the applications which require some added intelligence such as deciding on the origin of a whiskey, or the authenticity of an oil.

There is some debate about the best pretreatment of the voltage signals. Options include mean centring (subtraction of the mean of a column or row of the data from each value), double centring, standardization (division by the standard deviation of a column), and range scaling (causing the values to fall between 0 and 1). The choice of pretreatment depends on the nature of the information in the signals. Standardization, for example, tends to emphasize smaller signals (with commensurately smaller variance) over greater signals. The concentration dependence of the sensor response may be removed by dividing by the root mean square of the responses. This apparently drastic step improves the discrimination achieved by principal components analysis and cluster analysis and mimics the human nose, which is not efficient at perceiving the intensity of smells (Gardner 1991). A discussion of data processing with reference to the analysis of essential oils has been given by Hibbert (1996).

Moving on to the substantive part of data analysis, array sensors rarely follow well characterized functional forms, and certainly not independent

linear models as are the basis of methods such as principal components analysis. Despite this, linear methods such as principal components analysis (PCA) and principal components regression (PCR), partial least squares regression (PLS), discriminant analysis and cluster analysis (Gardner 1991; Auge *et al.* 1995; Stetter *et al.* 1993; Seemann *et al.* 1997; Barko *et al.* 1995; Sundgren *et al.* 1990) are found as often as the most prevalent of the non-linear methods, artificial neural networks (Auge *et al.* 1995; Seemann *et al.* 1997; Sundgren *et al.* 1990; Nakamoto *et al.* 1993; Singh *et al.* 1996).

However, except for very simple cases PLS or PCR does not do well against either non-linear PLS or artificial neural networks (Stetter *et al.* 1993) because of the non-linear nature of sensor data. Other methods published include K-nearest neighbour (KNN) classification (Barko *et al.* 1995; Sundgren *et al.* 1990), feature extraction methods (Seemann *et al.* 1997; di Natale *et al.* 1995*a*) and vector representation (Weimar *et al.* 1990).

Artificial neural networks have been shown to work well for complex, non-linear classifications. No model is assumed, the calibration data is presented to the ANN, and an internal model is built. ANNs are particularly powerful when used for classification, for example distinguishing among beers, wines, cheeses, and other foods. Most commonly published cases use a feed forward net with back propagation for training.

ANNs are powerful, but must be used in context. Calibration, discussed in general above, must use standards that span all possible combinations of input parameters and all classification outcomes. Back propagation training is computer intensive, which is a drawback if recalibration, because of drifting sensors or changes in the system being monitored, is necessary (Nakamoto *et al.* 1993). The requirements of re-training are also not well understood. Pruning, the removal of sensors (i.e. ignoring a sensor which does not contribute to the classification model) or the number of hidden neurons in the network, is necessary to avoid overtraining (Seemann *et al.* 1997). That this problem (development of the network until the fit to the training set is excellent but predictive ability is lost) is perceived in the development of array sensors may reflect the tendency to use too few training sets.

The presentation of each training set to the sensor array is an experiment for which a standard has to be provided, so it is not as easy to obtain comprehensive training sets, as, for example, for vision-based systems. Parsimony in the number of sensors recalls the theoretical treatment of Muller (1991), and underlines the fact that data is no use unless it contains extractable information. Recently the speed of self organizing maps (di Natale *et al.* 1995*b*) and self organizing adaptive resonance networks (Gardner *et al.* 1996) have been shown to be useful in analysing data from sensor arrays.

The treatment of complex data from a sensor array will probably fall into the category of 'intelligent data analysis'. However, of the many papers published with different sensors and different methods of data analysis, the majority are laboratory-based studies on contrived or extremely restricted sets of data. Sales of commercial electronic nose instruments suggest some are deployed in industry, but anecdotal evidence implies that there have been no great success stories of their use in genuinely intelligent situations. The problem lies as much with the chemistry of the sensors as with the data

analysis, but this field remains a challenge for those who would claim to have produced intelligent data analysis methods.

12. Conclusion

Intelligent data analysis does exist. Modern instrumentation provides such a great quantity of data that, even at an early stage, intelligence must be shown to manipulate and analyse them. Future intelligent instruments and autonomous intelligent machines will require sophisticated data analysis. This book aims to describe where mainstream intelligent data analysis is now, and outline trends.

References

Atema, J. (1995). Chemical signals in the marine environment: dispersal, detection and temporal signal analysis. *Proceedings of the National Academy of Sciences USA*, **94**, 62–6.

Auge, J., Hauptmann, P., Hartmann, J., Roesler, S., and Lucklum, R. (1995). Versatile microcontrolled gas sensor array system using the quartz microbalance principle and pattern recognition methods. *Sensors and Actuators B*, **26**, 181–6.

Barko, G., Papp, B., and Hlavay, J. (1995). Application of pattern recognition and piezoelectric sensor array for the detection of organic compounds. *Talanta*, **42**, 475–82.

Bell, G.A. (1996). Molecular mechanisms of olfactory perception: their potential for future technologies. *Trends in Food Science and Technology*, **7**, 425–31.

Brignell, J. (1996). Introduction to feature on intelligent instruments. *Measurement and Control*, **29**, 164.

Chiba, A. (1990). Development of the TGS gas sensor. *Chemical Sensor Technology*, **2**, 1–18.

de Bièvre, P. and Taylor, P.D.P. (1997). Traceability to the SI of amount-of-substance measurements: from ignoring to realizing a chemist's view. *Metrologia*, **34**, 67–75.

Deveza, R., Russell, A., Thiel, D.V., and Mackay-Sim, A. (1994). Odor sensing for robot guidance. *The International Journal of Robotics Research*, **13**, 232–9.

Dickinson, T.A., White, J., Kauer, J.S., and Walt, D.R. (1996). A chemical detecting system based on a cross-reactive optical sensor array, *Nature*, **382**, 697–700.

Di Natale, C., Davide, F.A.M., D'Amico, A., Sberveglieri, G., Nelli, P., Faglia, G., *et al.* (1995*a*). Complex chemical pattern recognition with sensor array: the discrimination of vintage years of wine. *Sensors and Actuators B*, **25**, 801–4.

Di Natale, C., Davide, F.A.M., and D'Amico, A. (1995*b*). A self-organizing system for pattern classification: time varying statistics and sensor drift effects. *Sensors and Actuators B*, **27**, 237–41.

Gardner, J.W. (1991). Detection of vapors and odors from a multisensor array using pattern recognition. Part 1. Principal component and cluster analysis. *Sensors and Actuators B*, **4**, 109–15.

Gardner, J.W. and Bartlett, P.N. (1994). A brief history of electronic noses. *Sensors and Actuators B*, **18**, 211–20.

Gardner, J.W., Hines, E.L., and Pang, C. (1996). Detection of vapours and odours from a multisensor array using pattern recognition: self-organizing Adaptive Resonance Techniques. *Measurement and Control*, **29**, 172–8.

Green, J.M. (1996). A practical guide to analytical method validation. *Analytical Chemistry*, **68**, 305A–309A.

Haddad, P.R. and Jackson, P.E. (1990). *Ion chromatography, principles and applications*. Journal of Chromatography Library, Vol. 46. Elsevier, Amsterdam.

Hibbert, D.B. (1993). *Introduction to electrochemistry*. Macmillan Press, London.

Hibbert, D.B. (1996). Chemometric analysis of data from essential oils. In *Modern methods of plant analysis. Vol. 19: Analysis of plant fragrance* (ed. H.F. Linskens and J.F. Jackson), pp. 119–40. Springer, Berlin.

Hierlemann, A., Weimar, U., Kraus, G., Schweizer-Berberich, M., and Goepel, W. (1995). Polymer-based sensor arrays and multicomponent analysis for the detection of hazardous organic vapors in the environment. *Sensors and Actuators B*, **26**, 126–34.

Hivert, B., Hoummady, M., Henrioud, J.M., and Hauden, D. (1994). Feasibility of surface acoustic wave (SAW) sensor array processing with formal neural networks. *Sensors and Actuators B*, **19**, 645–8.

Hodgins, D. and Simmonds, D. (1995). The electronic nose and its application to the manufacture of food products. *Journal of Automation in Chemistry*, **17**, 179–85.

Kaarls, R., and Quinn, T.J. (1997). The Comité Consultatif pour la quantité de matière: a brief review of its origin and present activities. *Metrologia*, **34**, 1–5.

Luttenbacher, D., Roth, S., and Robert, M. (1996). Application of object-oriented technology to intelligent sensor development. *Measurement*, **17**, 201–16.

Martens, H., and Naes, T. (1989). *Multivariate calibration*. Wiley, Chichester, New York.

Minsky, M. (1968). Matter, mind and models. In *Semantic information processing* (ed. M. Minsky), pp. 425–32. MIT Press, Cambridge, Mass.

Mulholland, M., Preston, P., Sammut, C., Hibbert, D.B., and Compton, P. (1993). An expert system for ion chromatography developed using machine learning and knowledge in context. In *Industrial and engineering applications of artificial intelligence and expert systems*, Proceedings of the 6th International Conference, Edinburgh, June 1993, (ed. P.W.H. Chung, G. Lovegrove, and M. Ali), pp. 258–67. Gordon and Breach Science Publishers, Switzerland.

Mulholland, M., Hibbert, D.B., Haddad P., and Sammut, C. (1995*a*). Application of the C4.5 classifier to building and expert system for ion chromatography. *Chemometrics and Intelligent Laboratory Systems*, **27**, 95–104.

Mulholland, M., Hibbert, D.B., Haddad, P.R., and Parslov, P. (1995*b*). A comparison of classification in artificial intelligence, induction versus neural networks. *Chemometrics and Intelligent Laboratory Systems*, **30**, 117–28.

Mulholland, M., Preston, P., Hibbert, D.B., Haddad, P.R., and Compton, P. (1996). Teaching a computer ion chromatography from a database of published methods. *Journal of Chromatography*, **739**, 15–24.

Muller, R (1991). High electronic selectivity obtainable with non-selective chemosensors. *Sensors and Actuators B*, **4**, 35–9.

Nakamoto, T., Fukuda, A., and Moriizumi, T. (1993). Perfume and flavor identification by odor-sensing system using quartz-resonator sensor array and neural-network pattern recognition. *Sensors and Actuators B*, **10**, 85–90.

Ramadan, Z., Mulholland, M., Hibbert, D.B., Preston, P., Compton, P., and Haddad, P.R. (1998*a*). Towards an expert system in ion-exclusion chromatography by means of multiple classification ripple down rules, *Journal of Chromatography*, **804**, 29–35.

Ramadan, Z., Mulholland, M., and Hibbert, D.B. (1998*b*). Classification of detectors for ion chromatography using PCA and LDA methods. *Chemometrics and Intelligent Laboratory Systems*.

Riviere, J.-M., Bayart, M., Thiriet, J.-M., Boras, A., and Robert, M. (1996). Intelligent instruments: some modelling approaches. *Measurement and Control*, **29**, 179–86.

Seemann, J., Rapp, F.R., and Gauglitz, G. (1997). Fresenius' Journal Analytical Chemistry, **359**, 100–6.

Sharaf, M.A., Illman, D.L., and Kowalski, B.R. (1986). *Chemometrics*. John Wiley and Sons, New York.

Singh, S., Hines, E.L., and Gardner, J.W. (1996). Fuzzy neural computing of coffee and tainted-water data from an electronic nose. *Sensors and Actuators B*, **30**, 185–90.

Sobolev, V. and Aumala, O. (1996). Metrological automatic support of measurement results in intelligent measurement systems. *Measurement*, **17**, 151–9.

Stetter, J.R., Findlay, M.W., Schroeder, K.M., Yue, C., and Penrose, W.R. (1993). Quality classification of grain using a sensor array and pattern recognition. *Analytica Chimica Acta*, **284**, 1–11.

Sundgren, H., Lundstroem, I., Winquist, F., Lukkari, I., Carlsson, R., and Wold, S. (1990). Evaluation of a multiple gas mixture with a simple MOSFET as sensor array and pattern recognition. *Sensors and Actuators B*, **2**, 115–23.

Taner, A.H. and White, N.M. (1996). Virtual instrumentation: a solution to the problem of design complexity in intelligent instruments. *Measurement and Control*, **29**, 165–71.

van Kampen, A.H.C., Ramadan, Z., Mulholland, M., Hibbert, D.B., and Buydens, L.M.C. (1997). Learning classification rules from an ion chromatography database using a genetic based classification system. *Analytica Chimica Acta*, **344**, 1–15.

Waldrop, M.M. (1987). Computers amplify Black Monday. *Science*, **238**, 602–4.

Weimar, U., Schierbaum, K.D., Goepel, W., and Kowalkowski, R. (1990). Pattern recognition methods for gas mixture analysis: application to sensor arrays based upon tin dioxide. *Sensors and Actuators B*, **1**, 93–6.

Whitby, B. (1988). *Artificial intelligence: a handbook of professionalism*. Ellis Horwood, Chichester.

2 Knowledge transfer: human experts to expert systems

Sharbari Lahiri and Martin J. Stillman

1 Introduction

At a meeting in 1956 at Dartmouth College, Marvin Minsky, John McCarthy, Nathaniel Rochester, and Claude Shannon gave birth to the term 'artificial intelligence' and announced the first computer program, the Logic Theorist, in artificial intelligence (AI). This program was developed by Allen Newell and Herbert Simon of Carnegie Mellon and J.C. Shaw of the Rand Corp. The research of Newell, Shaw, and Simon on the Logic Theorist, a chess playing program, and the General Problem Solver (GPS) dominated the first decade of AI from the mid-1950s to the mid-1960s. Their research in areas such as heuristic search, problem solving, planning, and knowledge representation remain important areas of AI and lead to the current generation of expert system programs.

In studying expert system applications in science, the hardware and software components serve only as tools to represent and apply scientific knowledge. The key to the application of AI techniques is knowledge coding. In this chapter we describe and illustrate implementation of techniques of knowledge coding that allow for greater use of expert systems. We begin with a discussion of the transfer of human expertise into an expert system, continuing with a discussion of the two important stages involved in the transfer of human expertise, namely (a) knowledge acquisition and (b) knowledge representation, and end by discussing our approach to building rule-based expert systems.

1.1 Simulation of human thought processes

GPS was one of the first programs created to simulate human thought processes. In the GPS:

- the problem was expressed as an external representation,
- a translator converted the problem into an internal representation, and
- problem solving techniques provided the solution by processing the internal representation.

The power of the GPS was based on the effectiveness of the problem solving techniques used and its generality was demonstrated by showing that the program could solve problems associated with different knowledge domains.

Problem solving was based on a heuristic search that was guided by a technique called 'means-end analysis'. In means-end analysis, the approach to analysis was based on the desired solution (end to be reached). The GPS consists of a group of methods capable of solving one or more problems. A 'big switch' model of generality was proposed if there was a need for a diagnostic routine to relate a given problem to a particular method.

However, the concept of generality was found to give rise to systems that were inefficient in terms of performance. The development of specific knowledge based systems began in 1965 when work started on DENDRAL, a system to interpret mass spectral fragmentation patterns. In 1973, researchers at Stanford University reported their results for MYCIN, a knowledge based system that was used to offer advice on possible treatments of bacterial infections in blood. In 1974, PROSPECTOR, an expert system to aid geologists in ore exploration was developed at the Stanford Research Institute. DENDRAL, MYCIN, and PROSPECTOR are landmarks in the history of the application of artificial intelligence techniques to complicated real world problems because each program was designed to solve a specific problem. DENDRAL and MYCIN showed that knowledge based systems offered high levels of performance because the emphasis was on *knowledge accumulation in a particular area*. The success of these expert systems led to the development and expansion of this new discipline.

Expert systems and neural networks have since been used extensively in chemistry. Typical applications include (i) interpretation of possible chemical structures from spectroscopic data, (ii) choice of optimal parameters for instrument operation, (iii) selection of an appropriate method of analysis, and (iv) diagnostic systems in which causes of chemical problems are inferred from a list of symptoms (most of the references given at the end of this chapter illustrate such applications). Each of these expert systems makes use of heuristics and inference mechanisms to represent and use knowledge belonging to a particular domain. Table 2.1 lists various expert system projects in the authors' laboratory where the key research has been in the knowledge coding and user interface steps. The emphasis on diagnostic expert systems for analytical chemistry was chosen because the knowledge domains are so well known that many experts are available, making knowledge acquisition easier. In addition, once the system is complete, there are many who can verify the accuracy of the decision-making that is based on the coded knowledge base.

Although similar in context, procedural artificial neural networks are composed of input and output units that simulate the computational aspects of the human brain. Neural networks have been used for spectral interpretation, recognition of flow injection patterns, qualitative and semiquantitative analysis in ICP-AES, and for modeling and prediction in multicomponent analysis. Unlike expert systems, neural networks are considered to handle 'fuzziness' in the data and information well.

Expert systems can be designed to handle modifications in knowledge through revisions of the knowledge base. However, this can be difficult, therefore it is important to make the construction, expansion, and modification of the knowledge base of an expert system efficient and user friendly through the availability of appropriate tools.

Table 2.1 A summary of the applications developed in the Stillman Laboratory at the U.W.O. (Reproduced with permission from Zhu and Stillman 1996.)

Program (personnel)	Module	Software Tool	Description
EAshell (G. Huang)	EAengine	C	A Windows–based inference engine, accessible by DLL function call.
	TableGenerator	C, Excel	A tool kit for the knowledge acquisition process.
	RuleEditor	C	A batch processor to convert a filled KDM into a rule file (KBF).
AAexpert (S. Lahiri)	AAdiagnosis	KDS, EAshell, Visual Basic	A diagnostic expert system for the atomic absorption spectrometer.
	AAmethod	EAshell, Visual Basic	An expert system for method selection in flame AAS.
	AAcontrol	EAshell, C, Fortran	A control program for automated AAS analysis of trace metals.
GCexpert (H. Du)	GCdiagnosis	EAshell, C, Visual Basic	A diagnostic expert system for gas chromatography.
	GC–QC	EAshell, C	A module performing data analysis.
SPILLexpert (Q. Zhu)	ACselect	Quick Basic, KDS	A module for selection of proper analytical methods based on the matrix, concentration range, and detection limit required.
	ERexpert	EAshell, Access, Visual Basic	A program using both an internal database and an expert system module to advise on the best response to emergency chemical spill accidents.
	ACmethod	EAshell, KDS, Access, Visual Basic	A updated version of ACselect comprising a database of methods and an expert system component.
GCMSdiagnosis (Q. Zhu)	QISMSdiagnosis	EAshell, Visual Basic	A diagnostic expert system for GC tandem mass spectrometer using the quadrupole ion storage device.
	SPECview	Visual Basic	A module directly access the Varian Saturn series of GCMS data
	DIAGplatform	Visual Basic	A module automatically calculates the conversion efficiency of a GC–MS–MS process.

2. Building an expert system: the problem domain and design of the prototype

The completeness of the description of the problem domain is the most important criterion in the successful development of expert systems. The

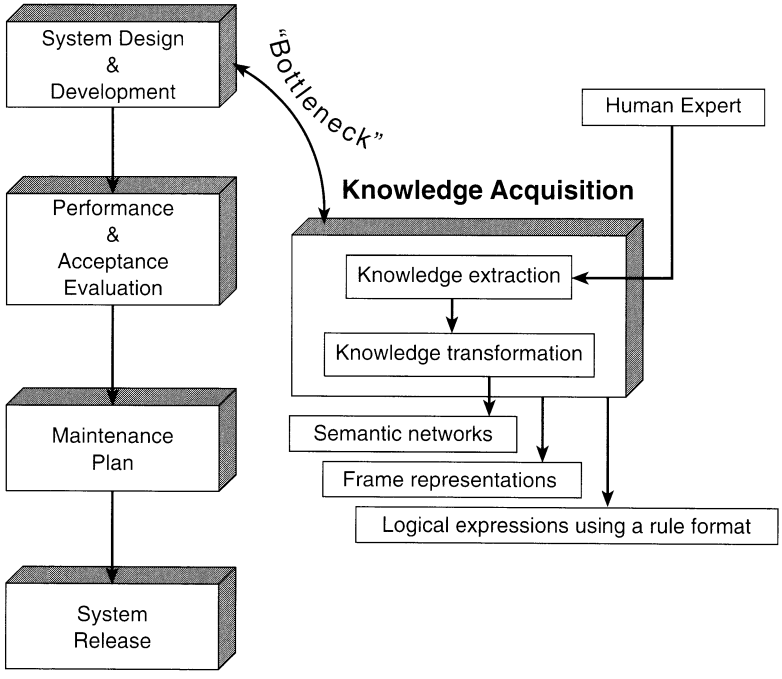

Fig. 2.1 The stages involved in the construction of an expert system application. Knowledge acquisition is a particularly important step that can be the bottleneck for subsequent development work. Reproduced with permission from Zhu and Stillman 1996.

crucial factors that determine whether an expert system will be able to solve a particular problem are: the nature of the problem, the availability of human expertise, and the ability to analyse the expertise and the problem in such a way that the knowledge can be coded into the knowledge base (Jackson 1990).

Full scale development of an expert system should be preceded by development of a prototype that can be used to indicate the appropriateness of using an expert system to solve the problem (Waterman 1986). Prototype design requires that an overall plan for the development process be determined and individual steps outlined. Figure 2.1 shows the stages commonly required. In many situations, knowledge acquisition proves to be the most difficult task to complete because of the interaction with experts who are not part of the development team — instead they are co-opted to provide their expertise. Personnel scheduling and cooperation of the various experts in allowing sometimes naive discussions of their expertise combine to reduce productivity at this crucial step.

The following example shows the relationship between the problem domain and the design of the knowledge base. Planning for the coding of the knowledge base requires that the domain knowledge be very clearly delineated. Figure 2.2, taken from Zhu and Stillman (1995*a*), shows how the deci-

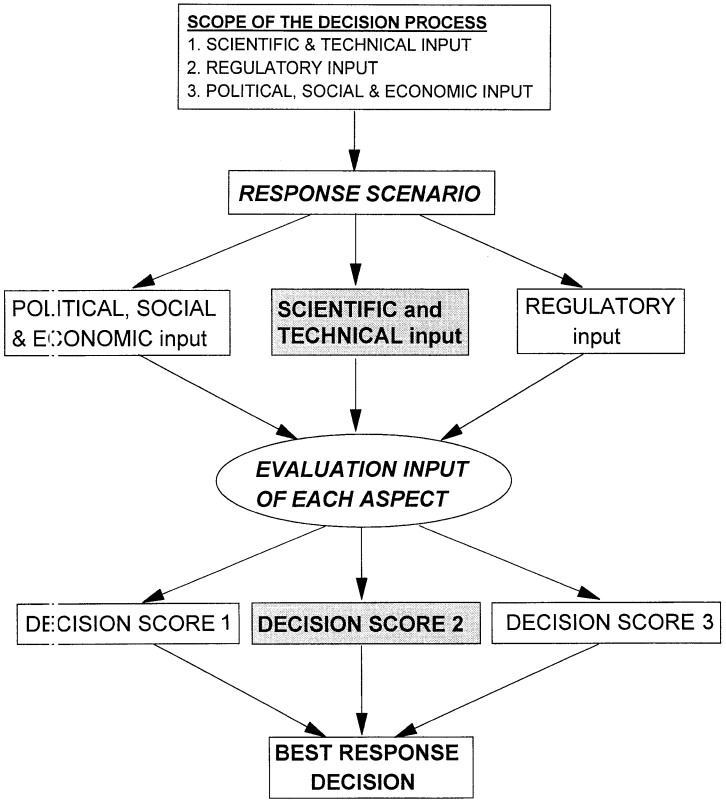

Fig. 2.2 Description of the different aspects involved in the cleanup processes for a chemical spill accident. Three factors need to be considered: (i) scientific and technological responses, (ii) regulatory constraints, and (iii) political, economic, and social demands. Reproduced with permission from Zhu and Stillman 1995*a*.

sion making process required following a chemical spill must be broken down into a series of subdomains.

For each area different decision-making processes are involved and each subdomain must be considered independently. A significant part of the problem domain is concerned with the quality of the advice given following an emergency. Providing advice on the correct action to be taken following a spill is complicated by the random nature of the emergency, yet much can be included in the knowledge base to reduce the difficulty in selecting a small number of decisions to offer to the user. The figure shows how the chain of inference must be completed in parallel and that each of these conclusions must be weighed to obtain the best decision.

Figure 2.3 shows the sequence of tasks required as part of the scientific and technical assessment of the response to a spill. The domain that was described for this project was concerned with the reaction by all levels of response team to a random catastrophic event, here a chemical spill in a public place. ERexpert is a program designed to provide the correct notification and assessment information from data provided by the response

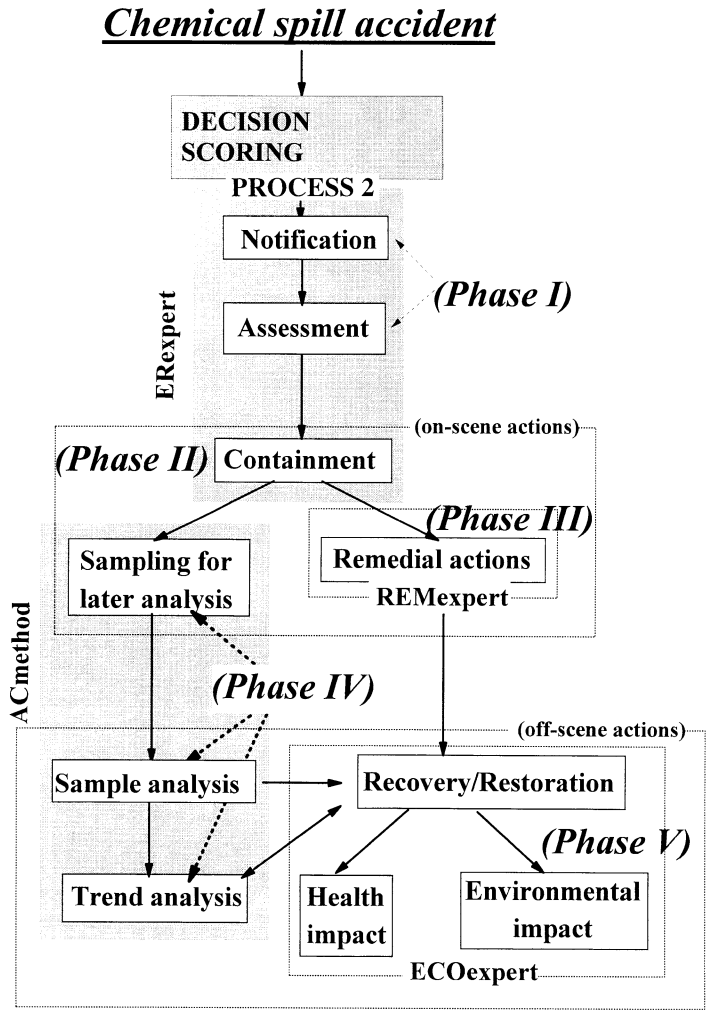

Fig. 2.3 The decision making hierarchy applied in response to a chemical spill. Five phases of the response are outlined. Reproduced with permission from Zhu and Stillman 1995*a*.

team. This part of the response must be rapid and so the program uses a combination of databases of known chemicals and rules for unidentified chemicals. The steps outlined as Phase III in Fig. 2.3 provide advice on the best containment and remedial action to be carried out at the scene. Phases IV and V involve the follow-up work to restore the condition of the area. ERexpert requires extensive data on the properties of chemicals (for example, so that the containment advice can be determined for different environmental conditions, an isolated stream compared with an urban roadway). Figure 2.4 shows the design of the factbase used in ERexpert (Zhu and Stillman 1995*a*). Clearly, the knowledge domain for this problem is multi-faceted with areas of expertise from chemical identification to chemical properties to chemical containment in the natural environment.

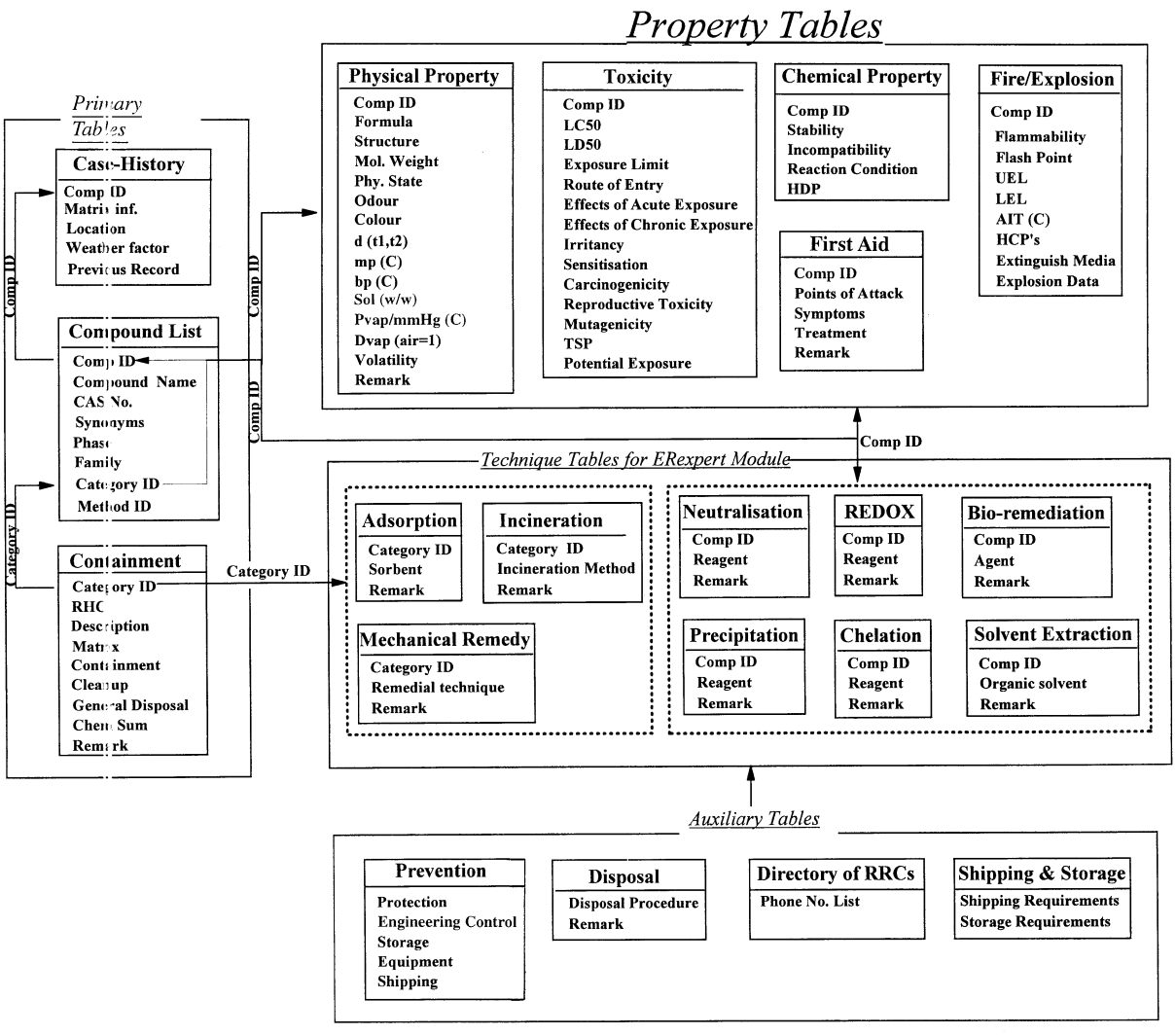

Fig. 2.4 Design of factbase used in ERexpert. Reproduced with permission from Zhu & Stillman 1995*a*.

2.1 Knowledge acquisition in expert systems

Once the problem domain has been well defined, the next step is to plan the acquisition step. Knowledge acquisition has been defined as the transfer and transformation of potential problem solving expertise from some knowledge source to a program (Jackson 1990). The different approaches to knowledge acquisition can be grouped as follows (Gruber 1990):

- Traditional knowledge engineering;
- Interactive knowledge acquisition tools; and
- Machine learning.

The traditional form of knowledge acquisition involves one or more experts describing their expertise and their line of reasoning in problem

solving. This step is often referred to as 'the bottleneck problem' in the generation of expert systems (Fig. 2.1) because it is found that it is very difficult to acquire knowledge from experts. The information that is extracted is then transferred to a form usable by the computer. This approach has also been referred to as the basic model of knowledge engineering. The major problem associated with this basic model is the assumption that knowledge engineers are available who can successfully transfer the specialized knowledge.

The interactive model of knowledge acquisition involves the expert directly transferring knowledge into a knowledge base and the knowledge engineer then collaborating with the expert to validate the knowledge base. Often, the knowledge acquisition tools require that knowledge is represented in a form that is conceptually or practically difficult for the expert to provide. For example, the expert may find it hard to supply the values when the knowledge acquisition tool requires the expert to provide weighted links associating data with hypotheses. Repertory grid centered tools for knowledge acquisition have been used to build knowledge based systems (Boose 1990). In a repertory grid, also known as a rating grid, solutions, referred to as elements, are placed in the columns and the traits, referred to as constructs, are placed as rows of the grid. These tools interact with the expert and perform tasks such as interviewing the expert, testing, and refining the resultant knowledge base. AQUINAS, an expanded version of the Expertise Transfer System (ETS), can automatically generate production rules from rating grids, which can be reformatted for use in various expert system building tools.

The third approach to knowledge acquisition, machine learning, involves the production of rules from examples. The implementation of machine learning requires setting up an induction algorithm to transform examples into rules. Quinlan (1983) developed an inductive algorithm (the ID3 algorithm) which has been applied in commercially available expert system shells, e.g. the KDS (Knowledge Delivery System) shell.

2.2 Knowledge representation in expert systems

Knowledge representation can be grouped into four major categories

- Logic
- Semantic Networks
- Frames and
- Rule-based Systems.

2.2.1 Logic

First order logic (FOL) expresses features of deductive reasoning as propositions. Propositions are defined as statements that are either TRUE or FALSE. Complex expressions are generated by using a set of symbols called connectives. Commonly used connectives are AND, OR, NOT, and IMPLIES.

The drawback to representing knowledge using FOL is that propositions in the real world cannot always be expressed as TRUE or FALSE. Multi-valued logic like fuzzy logic has been developed to express uncertainty (Pavelin 1988).

2.2.2 Semantic networks

Quillian was the first to apply semantic network ideas to the field of natural language translation and understanding. He proposed an associational model of human memory, called semantic memory, which attempted to capture the meanings of words (similar to capabilities of human memory) and implement this meaning in a computer program. Winston's idea of structured descriptions was based on generalizations from examples, similar to representation of human thought processes. The problem of using semantic networks to represent knowledge arises from the fact that definitions of meanings, also known as concepts, are subjective and difficult to incorporate into a program.

A semantic network is defined as a labeled directed graph that consists of vertices and labeled arcs between vertices. Each vertex represents a concept which is also known as a word meaning. The arcs represent binary relations between concepts. Relations commonly used in a semantic network are:

A 'part-of' relation: a relation in which the first concept is a '*part of*' the second concept. For example, halogens are a '*part of*' the Periodic Table.
An 'is-a' relation: there are two types of the '*is a*' relation between concepts.

(i) The set inclusion relation: a relation in which a concept '*is a*' subclass of another concept. For example, a transition metal '*is a*' metal, and
(ii) The membership relation: a relation in which a concept '*is a*' member of a certain class of objects. For example, copper '*is a*' transition metal.

Figure 2.5 shows the representation of knowledge that is contained in the periodic table of elements in the form of a semantic network.

The *subset-of* and *member-of* links can be used to derive new information and may form the basis for an inference engine. The use of links in a reasoning mechanism called inheritance is explained in the following example.

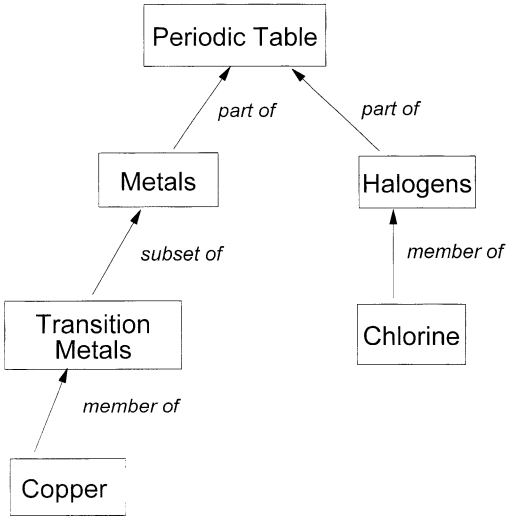

Fig. 2.5 Representation of knowledge using a semantic network. Reproduced with permission from Lahiri 1994.

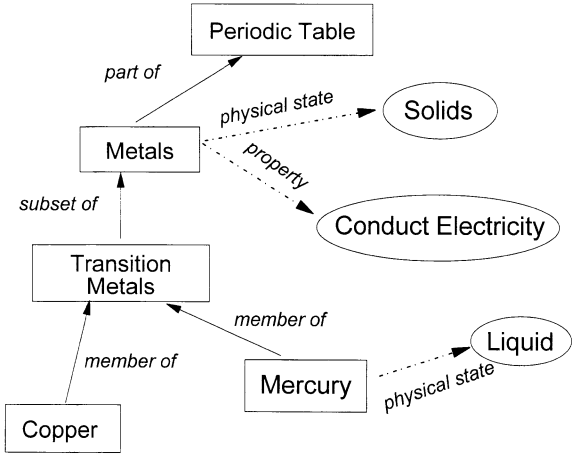

Figure 2.6 An exception in inheritance in a semantic network. Reproduced with permision From Lahiri 1994.

The statements represented in Figure 2.5 are:

Copper is a transition metal;
Transition metals are metals, and
Metals are elements of the Periodic Table.
From the above statements, the following statement can be derived.
Transition metals are elements of the Periodic Table.
The following statement can be derived using subset-of and member-of links:
Copper is a metal.

In an inheritance mechanism, the concept inherits properties of concepts higher in the semantic network through links. However, the hierarchical representation gives rise to problems in knowledge representation. Figure 2.6 depicts the problems associated with inheritance. The network shows that metals are solids and conduct electricity. Therefore, mercury, a metal, should inherit both properties of metals. Although mercury conducts electricity, it is not a solid. Flexibility has been introduced with regards to the inheritance of properties, by using knowledge representation in terms of frames (Lucas and Van der Gaag 1991).

2.2.3 Frames

In a frame-based representation, knowledge relevant to a concept is stored in entities called frames. A frame is defined as a network of nodes and relations organized in a hierarchy, where the topmost nodes represent general concepts and the lower nodes represent more specific instances of the concepts. This mode of knowledge representation is known as a frame hierarchy or frame taxonomy in which frames are represented by vertices and arcs denote '*is a*' links between two frames.

Class frames, also known as generic frames, represent knowledge concerning classes of objects. Knowledge concerning individual objects is repre-

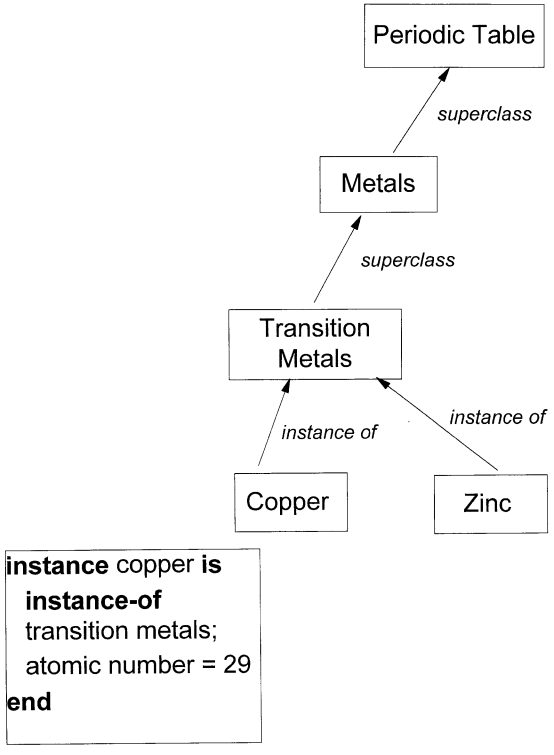

Fig. 2.7 Representation of knowledge using frames (tree-like taxonomy). Reproduced with permission from Lahiri 1994.

sented by instance frames. A frame indicates its relative position in a taxonomy by using two types of '*is a*' link.

(i) An instance-of link: a link between an instance frame and a class frame, and
(ii) A superclass link: a link between two class frames.

Figure 2.7 shows an example of frame taxonomy. Metals represent a superclass and copper is a specialization, a transition metal. Representation of knowledge in a frame takes the form:

```
instance copper is
instance-of transition metals;
group = 1 B;
atomic number = 29;
physical state = solid;
property = conducts electricity
end
```

Procedures present in frames are called demons and can be activated at a particular time during the manipulation of the frame. Commonly used demons are *if-needed*, *if-added*, and *if-removed*. Demons can also be used in knowledge based systems that use frames and production rules. Knowledge

in CENTAUR, a LISP-based expert system used to assist in the treatment of pulmonary diseases, is represented in the form of frames and production rules (Lucas and Van der Gaag 1991). It has been suggested that descriptive knowledge can be successfully represented in a frame-based system.

2.2.4 Rule-based systems

The concept of using rules to represent knowledge was introduced by researchers on the DENDRAL project. Rules represent knowledge in the form of IF — THEN statements. The IF part contains the premise of the rule and the THEN part contains the action or conclusion of the rule.

Example: IF pH of a solution = 3.0 THEN the solution is acidic.

The premise of a rule is a Boolean expression that must be satisfied for the rule to be executed (or fired). The conclusion of a rule can either be a list of commands to be carried out when the rule fires or be evaluated to true when the premise does. The set of rules that describe a particular knowledge domain is referred to as the rule base. In an expert system, the rule base along with facts associated with the knowledge domain forms the knowledge base.

Rules that are grouped in sub-areas of the problem domain are referred to as well-written rules. Well-written rules are said to be transparent in that the developer of the rule base is able to see through the syntax to the meaning. It is easy to modify knowledge in such a rule-based system because one part of the rule base can be changed without affecting the other parts of the rule base. Well-written rules should have the following features:

- **Organization of rules**: the maintainability of a rule based system improves if rules that have the same conclusion are grouped together;
- **Ordering of rules**: higher performance is achieved for an expert system when, in a particular group, rules are placed in order of most likely to least likely, and
- **Sequence of rules in a rule base**: the order of rules in a group should be based on the primary inference strategy used by the inference engine.

The drawback of a rule-based system is that it cannot represent structural knowledge. In a rule-based representation, it is not possible to represent knowledge describing a particular entity in the form of the clusters which are the characteristic feature of frame-based systems.

3. Encoding knowledge: a case history approach

There are two approaches in the implementation of a rule-based representation of knowledge: (i) the knowledge base can be designed for direct input of knowledge in the form of rules, and (ii) the system can be designed to generate rules from the available chemical knowledge. The maintainability of the knowledge base is important and adding knowledge to a system that requires input as rules is not an easy task. Therefore, in our work, we have opted for the second approach; that is to build rule-based expert systems based on the automated rule generation from case histories.

Table 2.2 The knowledge domain described in terms of case histories. (Reproduced with permission from Lahiri 1994.)

Knowledge domain	Observables	Action/Advice/Solution
Contamination_Blockage	The flame has a ragged appearance	The burner slot may be partially blocked
Contamination_Blockage	Occasional pulse observed from the absorption profile	Contaminated spray chamber
Contamination_Blockage; Solution_Problem	Lower than expected absorbance values	The burner slot may be partially blocked There may be contamination in the spray chamber The solution may be too viscous
Solution_Problem	Long rise time observed from the absorption profile	The solution is too viscous

We treat the chemical knowledge as facts, and we are more able to modify facts rather than a rule-based structure for problems in the chemical domain. We first described the design and implementation of a rule-based expert system to diagnose problems that arise during analysis by atomic absorption spectrometry (Lahiri and Stillman 1992). In this scheme, knowledge is represented as a matrix of observables (or symptoms) in rows and conclusions (or causes) in columns. Connections established the true–false relationships. Knowledge belonging to the same sub-domain is grouped in the order of most likely to least likely observables. This mode of depiction greatly aids in the verification, portability, and expansion of the knowledge base. The following example shows how the case history approach can be used to give generate rules (Table 2.2).

In the next step, the case history knowledge is transformed into the Knowledge Domain Table (Table 2.3). The connection between observables and conclusions can occur in several cells, for example row 2 in Table 2.3. When the input of knowledge is completed, rules are created using in-house software (Table 2.4).

4. Examples of expert systems that implement the case history approach

Over the last 10 years the Stillman group has developed the use of truth tables as a means of coding heuristic knowledge in terms of case histories. Figure 2.8 shows the sequence of coding knowledge in this manner.

Table 2.3 Representation of chemical knowledge in the form of a Knowledge Domain Table. (Reproduced with permission from Lahiri 1994.)

	Burner slot is blocked	Contaminated spray chamber	Check viscosity of solution
The flame has a ragged appearance	T		
Lower than expected absorbance values	T	T	T
Long rise time observed from the absorption profile			T
Occasional pulse observed from the absorption profile		T	

Table 2.4 Rules generated from the chemical knowledge present in the knowledge table. (Reproduced with permission from Lahiri 1994.)

Rule 1. IF The flame has a ragged appearance = True AND Lower than expected absorbance values = True THEN Contamn Blockage = Burner slot is blocked.

Rule 2. IF Lower than expected absorbance values = True AND Occasional pulse observed from the absorption profile = True THEN Contamn_Blockage = Contaminated spray chamber.

Rule 3. IF Lower than expected absorbance values = True AND Long rise time observed from the absorption profile = True THEN Soln_Problem = Check viscosity of the solution.

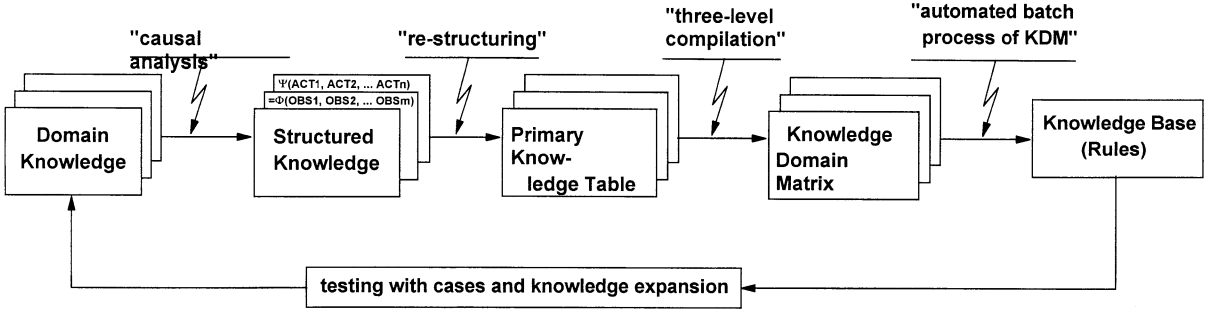

Fig. 2.8 The steps involved in the use of the Knowledge Domain Matrix to process heuristic knowledge that describes a defined problem domain. Reproduced with permission from Zhu and Stillman 1995*b*.

Following the assembly of the knowledge through knowledge acquisition from one or more experts causal analysis is carried out that relates observables to actions that are to be part of the decision. The sequence carried out involves providing observables that define the condition to be assessed and using more and more distinct observables to narrow the choice of conclusion. A series of compilation steps is carried out to ensure that each conclusion is identified by a unique set of observables, and that ambiguities are not inserted during such maintenance. Finally, the Knowledge Domain Matrix (KDM) is processed to generate a rule base. This last step can be automated as all ambiguities have been resolved in the previous steps. The rule base then can be prepared in any format, thus allowing the knowledge to be transferred from system to system. In fact, the KDM is an ideal depository of expertise because each condition and conclusion and the connecting logic (true or false) can be read using a standard spreadsheet program (EXCEL) as a viewer. This provides valuable portability as the expert can modify the matrix without any prior knowledge of the expert system shell.

The portability of the knowledge also means that experts world-wide can comment on the connections tabulated between observable or conditions and conclusions or decisions without access to the expert system. Indeed, knowledge engineers can process the KDM to provide rules in the form best suited to local expert system shells through filter programs that can allow many different expert system shells to use the same knowledge base.

Figure 2.9 illustrates how the truth table, as envisaged in the discussion above, rapidly becomes a complex database of knowledge. Figure 2.10 shows the complexity that arises when the similarities in observables or conditions require a large number of connections to provide unambiguous selection of the conclusion.

When two or more domains overlap and with increased complexity within a single domain, a three-dimensional structure for the KDM is required. Here conclusions from one level become conditions of another. A straightforward view of such a knowledge base is shown in Figure 2.10. The three layers can represent different components required in the decision making. With the appropriate user interface the layers do not need to be interconnected in the sequential manner shown in Figure 2.10.

We have developed expert systems to solve a number of chemical problems, from design of automated and unattended instrument analyses to emergency response to chemical spills. We have emphasized the importance of a universal means of converting chemical knowledge into rules that can be used by an expert system. The prototype AAexpert deals with automated analyses of metals by flame atomic absorption spectrometry (FAAS) (Browett and Stillman 1989; Browett *et al.* 1989; Lahiri and Stillman 1992; Lahiri *et al.* 1994) We have also implemented an expert system, GCDiagnosis, for diagnosing gas chromatographic data. The domain knowledge in these expert systems was acquired by studying the symptom to cause relationship by deliberately introducing faults in the analytical instruments. It was shown that error trapping by defining criteria and performance indices was an essential component involved in achievement of fully automated and unattended analyses. We have designed an expert system, ERexpert, that offers advice

	Volatile chemicals								Solid chemicals				Insoluble liquid/solid chemicals							
Conditions / **Conclusions**	1. Correct mechanic failure; close valve, clog leaks, shut down pump. To clean up, decide chemical type (Category ID), run KB again.	2. Avoid spark. Inert foam coverage with mist knock down.	3. Avoid spark. Cryogenic condensation, air dilution (fans/blowers).	4. Avoid spark. Encapsulation.	5. Inert foam coverage with mist knock down.	6. Cryogenic condensation, or air dilution using fans or blowers.	7. Encapsulation.	8. Solid chemicals are usually self contained. Recovery by shoveling or	9. Fluidized solids will be treated as liquids, concern is to prevent run off into watercourse/sewage, assign a liquid category ID to the target.	10. Treat as Category 2A/3A chemicals on dry surface.	11. Treat as fluidized Category 2A/3A chemicals.	12. No spark. Inert foam/plastic sheet, earthen dike, excavation, trench, pump/vacuum in auxiliary tank/sump, remove bottom, direct to paved	13. Earthen dike, trench, excavation, pump/vacuum to auxiliary tank/sump, remove bottom, direct to paved surface/sealant to slow penetration.	14. Inert foam/plastic sheet, excavation, earthen dike, trench to contain, pump/vacuum in auxiliary tank/sump, remove bottom, direct to paved	15. No spark contac. Earthen dike, trench, excavation, pump/vacuum in auxiliary tank/sump, remove bottom, direct to paved surface/sealant to	16. Avoid spark. Inert foam/plastic sheet,, foamed polyurethane, concrete dike to contain, pump/vacuum collect in auxiliary tank/sump, direct	17. Foamed polyurethane/concrete dike, pump/vacuum in auxiliary tank/sump, direct to paved surface/sealant. Or temporary burial.	18. Inert foam/plastic sheet, foamed polyurethane, concrete dike to contain, pump/vacuum in auxiliary tank/sump, direct to paved surface/sealant. Temporary burial to contain for later treatment.	19. Avoid spark contact. Foamed polyurethane, concrete dike to contain, pump/vacuum in auxiliary tank/sump, direct to paved surface/ surface sealant. Temporary burial to contain for later treatment.
1. Mechanical failure	T																			
2. Category 1A chemicals*		T	T	T																
3. Category 1B chemicals*					T	T	T													
4. Category 1C						t														
5. Category 1D chemicals			t																	
6. Sheltered area			T			T														
7. Calm area (air/water		T	T	T	T	T	T													
8. Limited spill size				T			T													
9. Category 2A chemicals*								T	T											
10. Category 3A chemicals*								t	t											
11. Spill happened on land								T	T	T	T	T	T	T	T	T	T	T	T	
12. Dry ground surface								T	F	T	F									
13. Category 2B chemicals*												T				T				
14. Category 2C chemicals*													T				T			
15. Category 2D chemicals*														T				T		
16. Category 2E chemicals*															T				T	
17. Category 3B chemicals*																				
18. Flat surface																T	T	T	T	
19. Soft surface/river bottom												T	T	T	T	F	F	F	F	
20. Flammable																				
21. Volatile																				
22. Liquid state										F	F									
23. Spill in watercourse																				
24. Category 4A chemicals*										T	T									
25. Category 4B										t	t									
26. Category 4C										t	t									
27. Category 4D chemicals*										t	t									
28. Category 4E chemicals*										t	t									
29. Category 4F chemicals*										t	t									
30. Natural barrier exists																				
31. Geographically feasible																				
32. Small removable volume																				
33. Clear area																				

Fig. 2.9 The Knowledge Domain Matrix for the second knowledge layer in ERexpert. 'T' indicates a unique connection whereas 't' indicates that this is a parallel case, a situation where a number of observables point to several conclusions. Reproduced with permission from Zhu and Stillman 1995*b*.

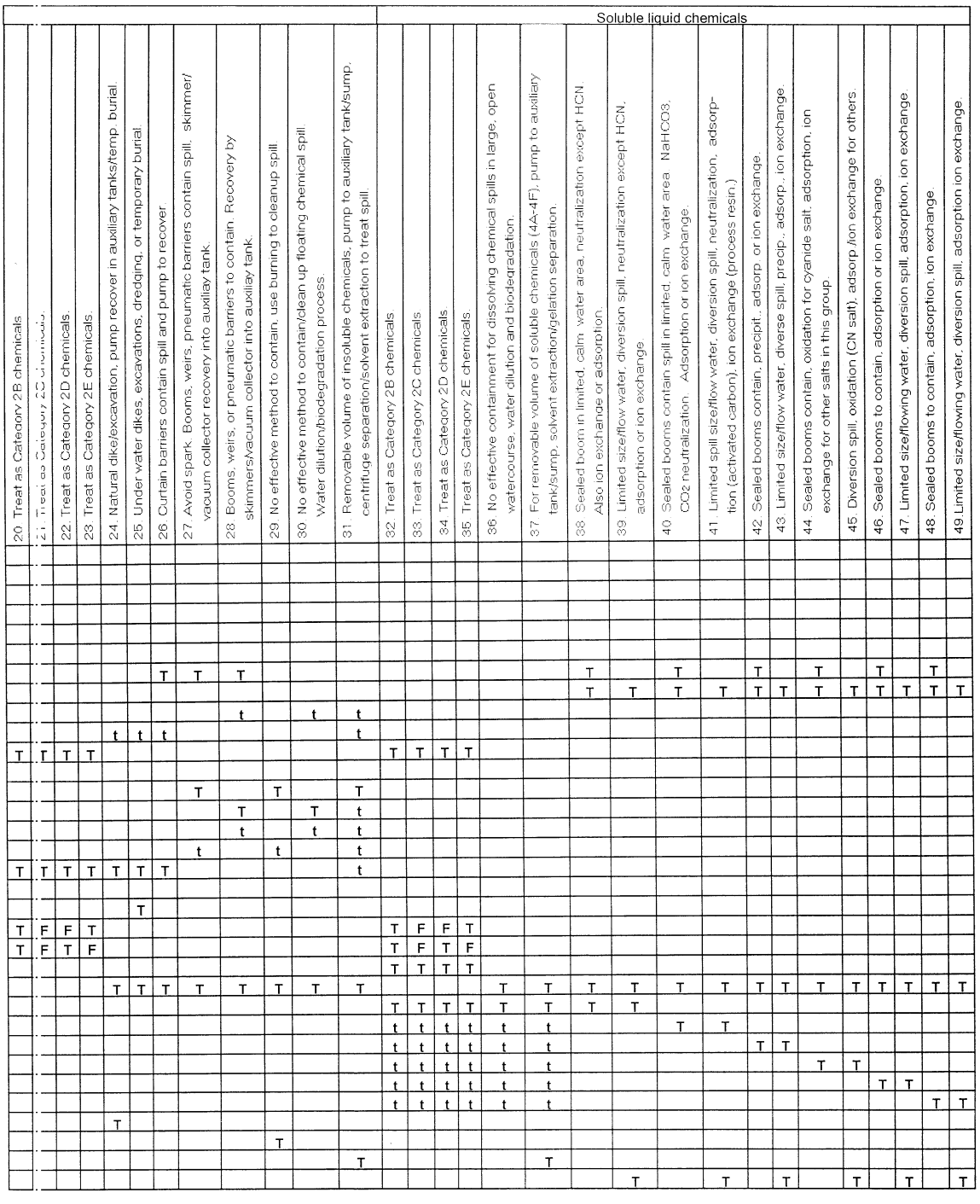

Fig. 2.9 *Continued*

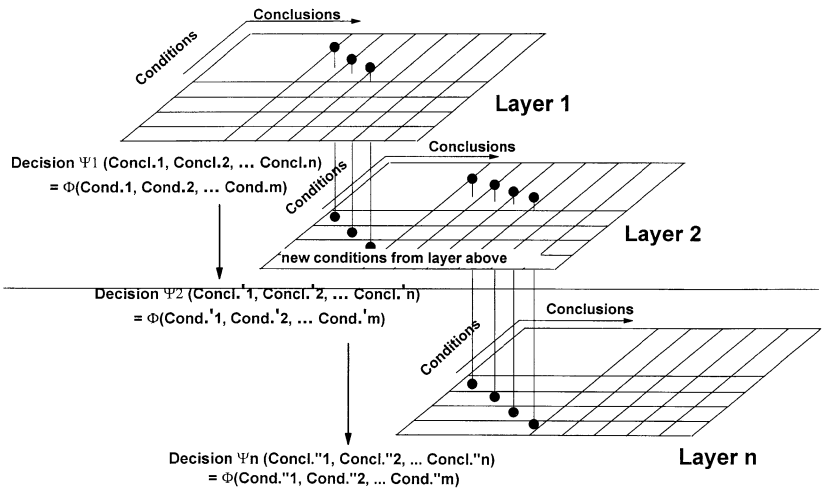

Fig. 2.10 A multilayer KDM designed to incorporate the complexity found in multistep decision making processes. Conclusions from one level are inserted as conditions in a second layer. Reproduced with permission from Zhu and Stillman 1996.

following an accidental chemical spill (Zhu and Stillman 1995*a*; Zhu and Stillman 1995*b*; Zhu and Stillman 1996). This uses a database structure that provides an effective archival method for organization of the elaborate factual information necessary in solving problems in the domain.

4.1 AAexpert: an example

This project is part of a large, multicomponent expert system named ACexpert, a system that is concerned with all aspects of instrumental analysis (Browett and Stillman 1989; Browett *et al.* 1989; Lahiri and Stillman 1992; Lahiri *et al.* 1994). A manual, non-automated model of the typical analytical laboratory was proposed, in which it was assumed that the analytical instrument and method to be used will be determined largely by regulatory agencies. The system is subdivided into a set of individual expert systems designed to perform specific tasks. The structure of AAexpert is shown in Figure 2.11.

AAexpert has been designed so that each module addresses a single, limited domain of expertise within the overall domain of analysis by Flame Atomic Absorption Spectroscopy. In this model, linked expert systems provided advice on each aspect of the analysis. On receipt of the sample the MANAGER's task is to consult with both the CUSTOMER and the REGULATORY AGENCY to determine the criteria to be used in the analysis. AAassurance is a quality assurance expert system that is used by the MANAGER and the ANALYST to assist in the execution of a laboratory quality assurance program. The ANALYST's role is supervision of the expert

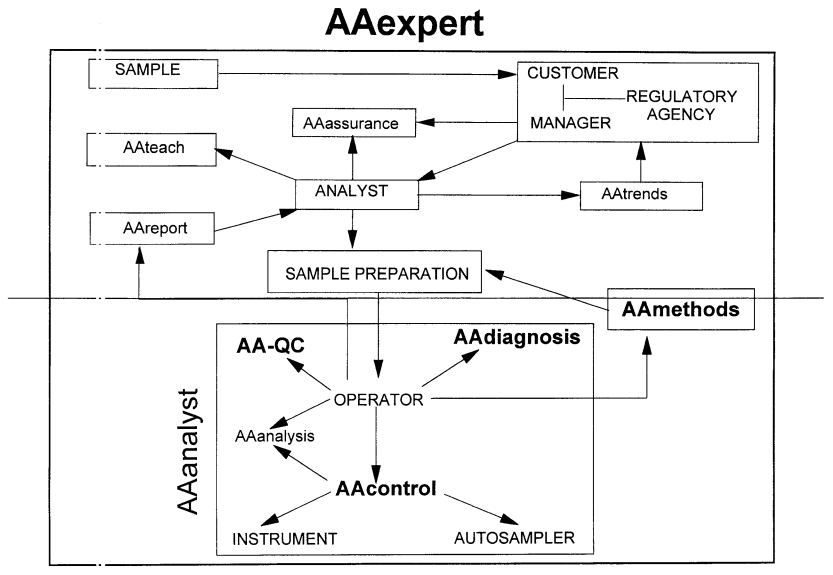

Fig. 2.11 The structure of AAexpert. Reproduced with permission from Browett and Stillman 1989.

system. AAanalyst, the process control and quality control expert system, will complete the required analyses using the modules: AAmethods for method selection, AAcontrol for sample scheduling and handling, AAdiagnosis for diagnosing faults associated with both instrumental operation and quality of data, and AA-QC for quality control. AAteach is an instruction expert system that uses simulation to give advice and examples on how to operate the instrument. The individual areas identified include method selection (AAmethods), control of the solution handling and measurement steps (AAcontrol), real-time assessment of analytical data (AA-Quality Control), and diagnosis of errors due to instrumental malfunction and interfering sample chemistry (AAdiagnosis). Each example involves a different application of expert system technology.

The critical steps involved in realization of real-time corrective control of analytical instruments are (i) obtaining measured data that reflect the instantaneous condition of the sample chamber and (ii) modeling the data in terms of analyte quality. Modeling the detector response implies understanding the physical and chemical processes which produce the analytical signal. The program dedicated to quality control made use of a rule base to identify commonly occurring problems associated with analysis by FAAS. Automated detection of problems can prompt the instrument to take remedial actions which may involve modification in sample preparation or stopping analysis. For example, the presence of out of range detector response is always associated with a concentrated sample. Automated signal interpretation and error handling can rectify the problem by diluting and reinjecting the sample. Visual detection of the error at the end of the batch analysis would have rendered the analysis of the remaining samples useless.

4.2 GCDiagnosis: an example

The knowledge base for the initial expert system for diagnosing problems associated with analysis by gas chromatography was published in 1994 (Du *et al.* 1995). We have also described a model that characterizes the physico-chemical processes of a gas chromatographic system with a flame ionization detector (Du and Stillman 1994). Causal analysis was used to generate and compile the knowledge base of GCDiagnosis. A standard test mixture was used to measure the peak parameters. The algorithm was based on information theory and chemical knowledge about the sample. GC-Assess program was developed to compare the peaks in the sample and reference chromatograms. This program was validated by using simulated data. It was found that even with severe distortion in peak retention times, GC–Assess was able to cope with variations in the simulated chromatogram. Relative Performance Indices were used to quantify instrument characteristics and describe the relationship between detector response and sample properties. The knowledge base was generated using an in-house EXCEL macro package that created unique rules free from syntax errors.

Before an instrument can carry out unattended, automated analyses, the controlling system must be able to interrupt or modify the operation in real time to correct instrument malfunctions and change sample measurement problems. However, if operator intervention is required, a diagnostic expert system can be used to act as an advisor. We have described the design of rule-based expert systems in which knowledge is represented as a matrix of observables and underlying causes. Although rule-based expert systems use the natural way of capturing expert knowledge, changes in the data involve a revision of the knowledge base. The major limitation of rule-based expert systems is its lack of learning ability.

4.3 ERexpert: an example

Zhu and Stillman have described the application of expert system technology to decision-making following a chemical spill. The key component in these studies was the combination of factual information in traditional databases with heuristic knowledge coded in a rule-base. The determination of the correct advice to give following a chemical spill is extremely complicated, requiring assessment of the chemicals involved, the environmental aspects of the spill, and legal and regulatory aspects of the cleanup. ERexpert was designed to accomplish the task of considering each of these components and provide advice for each phase of the response — from the immediate tasks following the spill to the remediation process necessary once the spill is contained. The structure of the knowledge base was discussed and the knowledge domain matrix used in the prototype described, shown in this chapter as Fig. 2.9.

Decision-making in response to any emergency is an ideal task for an expert system. The knowledge can be compiled and maintained over time between emergencies. Experts from around the world can participate in the compilation step and emergencies around the world can be used as case histories to validate the responses provided by the program. Finally, the challenges of making the very best decisions following such an accident require special-

ized tools — the expert system is one such tool that can be tuned to provide highly focused information to aid in the decision-making process.

5. Case histories to expert network

New work on the use of hybrid AI tools to diagnose problems associated with gas chromatographic analysis has been reported by researchers at Florida State University and Los Alamos National Laboratory (Levis *et al.* 1995; Elling *et al.* 1997). An expert network is a translated rule-based expert system that uses specialized nodes and functions uncommon to traditional artificial neural networks. This hybrid AI tool preserves the natural knowledge representation and explanation capability of rule-based programming, yet provides the learning capability of neural networks. The knowledge was represented as a matrix of case histories modified from that previously reported (Du *et al.* 1994; Du and Stillman 1995). The true–false representation was replaced by qualifiers such as Always, Usually, Sometimes, Infrequently, and Never to reflect uncertainty in the relationship between symptoms and causes. The expert network consisted of Symptom, Filter, Combination, and Fault nodes. The binary (e.g. irregular spikes, a symptom that is either present or absent) and fuzzy (e.g. tailing peaks, a symptom that occurs with varying degree of fault) outputs from algorithms which extract signal parameters from chromatograms are used as input for the Symptom node (Lahiri *et al.* 1996). A training algorithm that uses back-propagation of error was developed to optimize the weights of the connections between Filter and Combination nodes. The system was trained and tested using examples that were generated for specific faults. The untrained expert network was able to diagnose induced faults in 44 per cent of the examples, training on 25 per cent of the data produced an accuracy of 88 per cent. Incorporation of knowledge for multiple paths of reasoning increased the diagnostic capabilities of the expert network to 93 per cent accuracy. An expert network can be retrained when more examples are available or to account for small changes in instrument configuration and sample type. This improves the performance and robustness of the system.

6. Design of a model expert system

In this section we detail the criteria required in the development of a rule-based expert system. We propose that the design and implementation of an expert system be described in terms of phases, each phase containing one or more cycles. A typical design may involve three phases to arrive at the completed version of the program. The timeline picture (Fig. 2.12) shows phase I, which comprises two cycles; phase I is considered to be the most important part of the project. Phase II and phase III each contain one cycle.

At the end of the first three cycles, a test version of the expert system is released to users. Each task in the development cycle is modularized, this allows the prototype design and testing to be carried out at different stages of

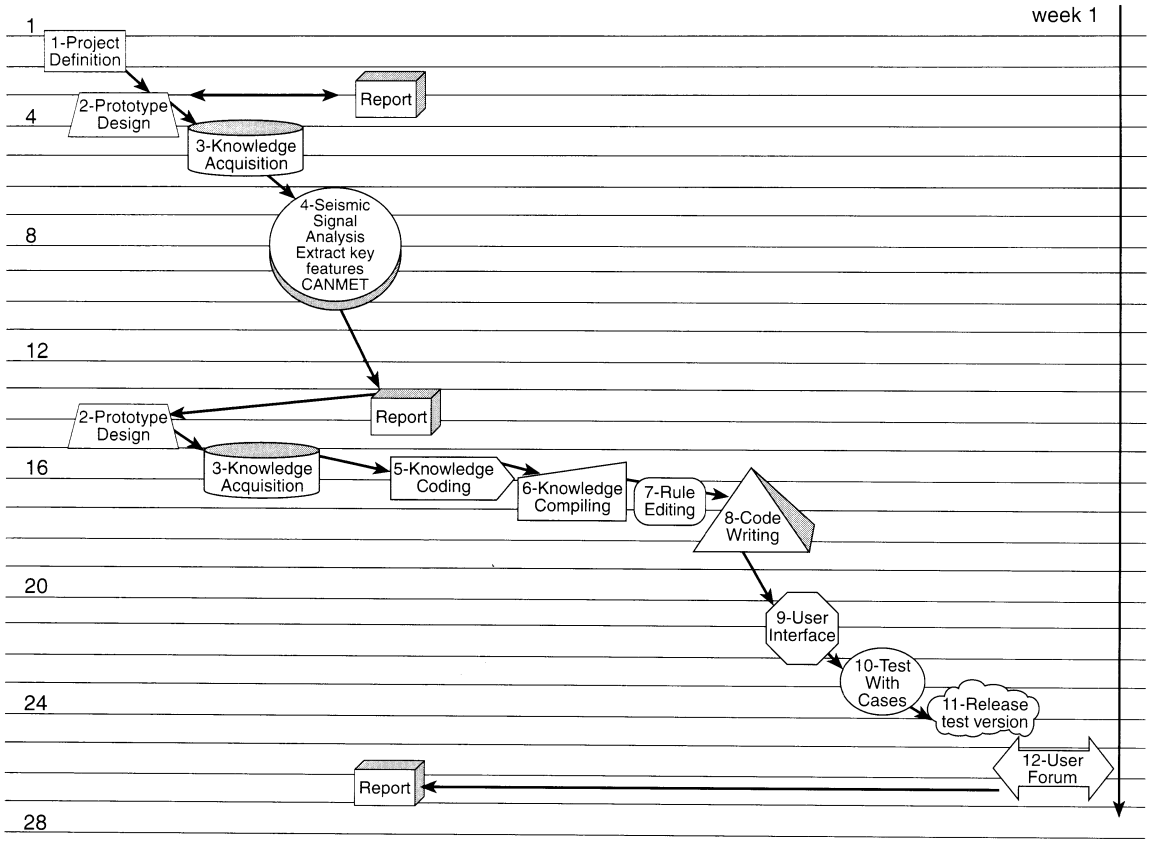

Fig. 2.12 Phase I in the design and implementation of a rule-based expert system. Lahiri and Stillman, unpublished work.

the project. Figure 2.12 shows the relationship between a number of specific tasks and their sequence in the development cycle. The key steps are:

Problem Definition: The expert, consultant, and knowledge engineer meet to discuss the problem and the problem-solving strategy. The knowledge engineer carries out an initial survey of the knowledge related to the problem domain and outlines the design of the expert system. At the end of this stage, the consultant meets with the expert to discuss the feasibility of the project and to finalize the contractual details.

Prototype Design: An initial structure of the expert system is delineated, the breadth and depth of the expertise is defined, and the applicability and extent of data analysis is assessed. The final prototype design is arrived at following the phase I of the project.

Knowledge Acquisition: Domain knowledge and heuristics used by a domain expert are captured in this step of the implementation. The expert is interviewed by the knowledge engineer and fills a knowledge template in an order of most likely to least likely observables. The knowledge template will include signal characteristics, description and the reasoning process. The knowledge engineer must be able to generalize the knowledge, extract the

concepts, and understand the trends and exceptions in the knowledge. Next, domain knowledge is classified into goals where relevant knowledge is presented as case histories. Coded knowledge is converted into rules.

Data analysis: This is one of the most important tasks of a project that deals with analyses involving signal interpretation. Increasing the number of defined signal features will help in the construction of an expert system that will be capable of providing more accurate advice to end users.

Code writing: This component in the development concerns programs needed to support the user interface, construction of the knowledge base, data analysis, and creation of the link between the user interface and inference engine. Code needs to be written to provide an explanation system that traces the line of reasoning and provides a detailed version of the remedial action.

User Forum: Because the success of an expert system is intimately linked to the acceptability of the program to users, potential users must be requested to provide feedback. This includes the ease of use (user interface), depth of knowledge, and quality of advice. User forum allows for exchange of ideas and incorporation of revisions. This is a critical part of the project as the expert system must provide the users with the information they require; the time taken in this stage will depend on the extent of the responses by the users.

Test with cases: The knowledge base and the quality of the advice given must be tested with conditions taken from the knowledge domain. This component provides testing of the knowledge base.

Reports: Provide milestones of the project.

The time required or the completion of stages involving *knowledge acquisition, release test version*, and *user forum* will be determined initially by the time allocated to the project developers by the experts and end users.

7. Conclusions

Expert systems are knowledge based computer programs that attempt to apply the experience of an expert in a particular area of knowledge. In studying expert system applications in chemistry, the hardware and software components serve only as tools to represent and apply the chemical knowledge. The role of expert systems is not to replace these scientists but to aid them with advice. Researchers interested in expert system applications in analytical chemistry have mainly focused on a single area of expertise, refined the knowledge base, and written prototypes.

There are two important stages involved in the transfer of human expertise into an expert system. These are: (i) encoding the chemical knowledge, and (ii) representing the chemical knowledge. There has been little work reported about how knowledge in general can be coded for use in an expert system. As computational tools become more available, the coding of chemical knowledge becomes more and more of a bottleneck. Once coded, the availability of different forms of knowledge representation introduces a knowledge transportation bottleneck because not only can rules not be readily transferred between systems, the knowledge encoded in a rule-based representation

cannot be used in other representations, for example knowledge coded in a rule-based system cannot be transferred to a frame-based system.

It is hoped that the acquisition of knowledge in the form of a matrix of conditions and conclusions will lead to a common method for encoding chemical knowledge for rule-based expert systems. This will allow such knowledge bases to be transported from one rule-based expert system tool to another.

8. Acknowledgments

We acknowledge the financial support of the Natural Sciences and Engineering Council of Canada and Imperial Oil Canada through the Imperial Oil Foundation.

References

Boose, J.H. (1990). In *The foundations in knowledge acquisition* (ed. J.H. Boose, B.R. Gaines) pp. 61–84 Academic Press, San Diego, USA.

Browett, W.R., Cox, T.A., and Stillman, M.J. (1989). Design of an expert system for automated metal analysis by atomic absorption spectrometry. *ACS Symposium Series*, **408**, 210–35.

Browett, W.R., and Stillman, M.J. (1989). Use of expert system shells in the design of ACexpert for automated atomic absorption spectrometry. *Progress in Analytical Spectroscopy*, **12**, 73–110.

Du, H., Huang, G., and Stillman, M.J. (1995). Automated knowledge base generation for the GCdiagnosis System. *Analytica Chimica Acta*, **324**, 85–101.

Du, H., Lahiri, S., Huang, G., and Stillman, M.J. (1994). Developing an expert system for diagnosis of problem GC data. *Analytica Chimica Acta*, **296**, 21–31.

Du, H. and Stillman, M.J. (1994). Knowledge acquisition for fault diagnosis in gas chromatography. *Analytica Chimica Acta*, **296**, 33–41.

Elling, J.W., Lahiri, S., Luck, J.P., Roberts, R.S., Hruska, S.I., Adair, K.L., Levis, A.P., Timpany, R.G., and Robinson, J.J. (1997). *Anal. Chem.*, **69**, 13, A409.

Gruber, T.R. (1990). In *The foundations in knowledge acquisition*, (ed. Boose, J.H., Gaines, B.R.), pp. 115–33, Academic Press, San Diego, USA.

Jackson, P. (1990). *Introduction to expert systems* (second edition), Ch. 1, pp. 3–14. Addison-Wesley Publishing Co., Reading, Massachusetts.

Lahiri, S. (1994) *Design and implementation of expert systems in analytical chemistry*. PhD Thesis, The University of Western Ontario.

Lahiri, S., Robert, R.S., and Elling, J.W. (1996). *J. Chrom. Sci.*, **34**, 505.

Lahiri, S., and Stillman, M.J. (1992). Expert systems: Diagnosing the cause of problem atomic absorption spectrometry data. *Analytical Chemistry*, **64**, 283A–291A.

Lahiri, S., Yuan, B., and Stillman, M.J. (1994). Toward totally automated analysis of trace metals by flame atomic absorption spectroscopy. *Anal. Chem.*, **66**, 2954–63.

Levis, A.P., Timpany, R.G., Austad, W.E., Elling, J.W., Ferguson, J.J., Klotter, D.A., and Hruska, S.I. (1995). *SPIE*, **2492**, 294.

Lindsay, R.K., Buchanan, B.G., Feigenbaum, E.A., and Lederberg, J. (1993) *Artif. Intell.*, **61**, 209.

Lucas, P., and Van Der Gaag, L. (1991). *Principles of expert systems* (Second edition). Ch. 4, pp. 173–252. Addison-Wesley Publishing Co., Wokingham, England.

Pavelin, C. (1988). In *Approaches to knowledge representation — an introduction*, (Ringland, G.A., and Duce, D.A., eds.) Ch. 2, pp. 13–43. Research Studies Press Ltd., Letchworth, Hertfordshire, England.

Quinlan, R.J. (1983). In *Machine learning — an artificial intelligence approach* (eds. Michalski R.S., Carbonell, J.G., and Mitchell, T.M.) Ch. 15, pp. 463–82. Tioga Publishing Co., Palo Alto, California.

Waterman D. (1986). *A guide to expert systems*, Ch. 12, pp. 135–41. Addison-Wesley Publishing Company, Reading, Massachusetts.

Zhu, Q. and Stillman, M.J. (1996). Expert systems and analytical chemistry: recent progress in the ACexpert project. *J. Chem. Inf. Comput. Sci*, **36**, 497–509.

3 The genetic algorithms, linkage learning, and scalable data mining

Hillol Kargupta, Eleonora Riva Sanseverino, Erik Johnson, and Samir Agrawal

1. Introduction

The Genetic Algorithms (GAs)(Holland 1975) are search algorithms motivated by the extra-cellular flow of evolutionary information through selection, crossover, and mutation. The GAs are typically applied to problems in which little knowledge is available about the quantitative properties of the objective function. The lack of quantitative information characterizing the search space for many real life search problems, the inherent suitability implementations, and apparent robustness against noisy data have made the GAs quite popular for solving large search, optimization, and machine learning related problems. In the recent past there has also been growing interest in the application of GAs for Knowledge Discovery and Data mining (KDD). The field of KDD deals with the problem of detecting patterns within large databases. Scalability (variations of performance quality with respect to growing problem difficulty, desired accuracy, reliability, computational resources) of KDD algorithms is an important issue, since large databases and high dimensional feature spaces are typical characteristics of the common KDD applications. Therefore the scalability of the GAs is likely to play a critical role in their success in large scale KDD applications.

This chapter focuses on the scalability issue of the GAs. It takes a detailed look at the fundamental underlying search processes in the GAs, points out some serious bottle-necks of frequently used simple GAs (Dejong 1975), presents a new scalable genetic algorithm, and outlines its application to a large-scale electrical power distribution network fault detection problem.

Section 2 reviews the related work on GA-based KDD. Sections 3 and 4 offer a perspective of black-box search/optimization (optimization in absence of adequate quantitative information regarding the search space) in the context of a general probabilistic and approximate framework. Section 5 presents a discussion on problem difficulty from this perspective. Section 6 discusses the simple GA in this context. Section 7 identifies a critical problem with the simple GA — lack of scalable mechanism for linkage learning (detection of appropriate relations among the search space members). Section 8 presents a new class of GAs capable of scalable linkage learning, called the

gene expression messy GAs. Sections 9 and 10 describe the power distribution network fault detection problem and present the experimental results. Finally, Section 11 concludes this chapter.

2. Data mining and the genetic algorithms

Knowledge Discovery and Data mining (KDD) is a fast growing field that deals with the research and practice of detecting patterns in data. While knowledge discovery is used to mean the complete process, including data pre-processing, extraction, analysis, and visualization, the phrase data mining usually represents the process of detecting data patterns using machine learning, statistics, and other techniques. The phrase 'data pattern' is usually used to mean relations among the data sets in a set theoretic sense. The relations are typically captured in terms of rules, similarly based subsets, and associations among the search space dimensions. Therefore, a data mining algorithm can also be viewed as a search for appropriate rules, similarities, or other kinds of associations. The GAs fit quite well into this application, and can be used for finding any of these pattern types. Apart from these, typically the data mining process requires feature selection, model optimization, and system identification techniques. The GAs are also suitable for such applications. There exists a growing body of literature on the application of the GAs to data analysis/mining problems. The following part of this section reviews some of these works.

Since machine learning algorithms find frequent applications in data mining, it is appropriate to review some of the early GA Based Machine Learning (GBML) systems. LS-1 (Smith 1980, 1983, 1984) is an example of one such early GBML system that used simple GA-like genetic operators to manipulate a population of production rules. They manipulated the representation at different levels of granularity reflecting the semantics of the representation showing that results of genetic algorithms still remained valid. In GABIL (DeJong, Spears, and Gordon 1993), Disjunctive Normal Form (DNF) concept descriptions are evolved using an LS-1 style approach. This work is aimed at a single class learning application. The goodness of a concept description is measured as the square of examples correctly classified. The COGIN approach developed elsewhere (Greene and Smith 1993, 1994) addresses multi-class problem domains introducing competition for coverage of training examples, encouraging the population to cooperatively solve the concept learning task. Each rule is a conjunction of attribute/value sets in binary coding. In this approach, the newly created rules using GA operators, together with the existing population of rules, are ranked in order of fitness and are inserted one by one in this rank order into the next generation of the population, provided they cover some example in the training set which has not already been covered by a previously inserted rule. Any such redundant rule is discarded. The population size thus changes dynamically according to the number of rules required to cover the entire set of training examples. Fitness is based on entropy measure, modified according to

classification accuracy. Both single point and uniform crossover have been used. Recombination is applied to the left hand sides of rules only. The right hand side of a rule is assigned to be the majority class found within training examples covered by the rule.

The REGAL system (Neri and Giordana 1995) uses a similar coverage-based approach for multi-concept learning. Each rule is evolved on its own and co-operation within population is encouraged through competition for coverage. This work introduced the *Universal Suffrage* selection operator, that selects rules providing larger coverage together. A parallel GA based approach (Cui, Fogarty, and Gammack 1993) is used for the identification of 'good' and 'bad' customers in credit-scoring applications. Solutions are evaluated using either a generic classification accuracy measure, or an application-specific measure of profitability. The results are then compared with other classification algorithms (Bayes, k-nearest neighbours, and ID3).

The GA-MINER system (Radcliffe 1995) is yet another effort for GA-based data mining. The author divides the data-mining problem into *undirected or pure data-mining, directed data-mining*, and *hypothesis testing and refinement*. Undirected data mining addresses the case where the system is left almost entirely unconstrained to find patterns in the data; whereas in *directed data-mining*, the user may specify some constraints. In *hypothesis testing and refinement*, the user poses some hypothesis and the system first evaluates the hypothesis and then tries to refine it. The GA-MINER system is supposed to be able to deal with all of these three levels of data-mining and uses a parallel genetic-algorithm based data-mining tool for data pattern discovery. One of its main components is the *pattern template*. It defines the general form of the patterns of *interest* to the user and reduces the search space to those patterns consistent with this form. The pattern interest is evaluated through statistical measures. In this work, the application concerns the identification of valuable patterns in large databases such as that obtained by the aggregation of census data and car sales data. This work also studies the scalability of the system with respect to increasing computational resources. In Punch *et al.* 1997, a feature selection and classification problem is dealt with using a GA combined with a K-nearest-neighbour algorithm to optimize classification looking for an optimal features weighting in order to efficiently accomplish the classification task. A GA-based data mining application has also been developed in (Bhargava and Jacobson 1997). This work used a GA to detect appropriate feature subsets and combined them to generate 'interesting' patterns from *Persian Gulf Syndrome* related data.

In this chapter we address at least one aspect of this big picture — the scalability issue. Although the volume of GA-based data mining application is increasing, more attention needs to be paid to scalability. This is an important issue in algorithm design that studies the performance of algorithms with respect to growing problem difficulty levels, desired accuracy, reliability, and the computational model. In this chapter we are primarily concerned with the scalability of GAs with respect to growing search space. We are interested in GAs that offer quality performance with amenable increase in computational cost as the size of the problem increases. In order to explore scalability we need to understand the fundamental search processes of a black-box search

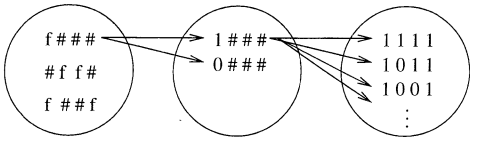

Fig. 3.1 BBO decomposition in relation, class and sample spaces. Note the similarity based equivalence relations. Here *f* denotes a position of equivalence and the [#] character matches with any binary value.

algorithm like the GA. The following section describes a probabilistic and approximate framework to do so.

3. Decomposing black-box search/optimization

Understanding the genetic algorithms (GAs) requires first understanding the foundation of non-enumerative black-box search/optimization (BBO) algorithms. The goal of a BBO algorithm is to find solution(s) from the search space that make the objective function value extreme beyond an acceptable criterion. Since for most interesting problems the search space is quite large, BBO algorithms, like the GAs, depend on non-enumerative search, which is actually an inductive process. This section makes a note of that and offers a decomposition of the underlying processes in a non-enumerative BBO algorithm using the Search Envisioned As Relation and Class Hierarchizing (SEARCH) framework.

The SEARCH framework (Kargupta 1995; Kargupta & Goldberg 1996) studies the fundamental issues in BBO by decomposing them into searches in (1) *relation*, (2) *class*, and (3) *sample* spaces (Fig. 3.1). SEARCH is based on the fact that induction is an essential part of non-enumerative BBO, since in the absence of any analytic information about the objective function structure, a BBO algorithm must guess based on the samples it takes from the search space. SEARCH also notes that induction is no better than table look up unless we restrict the scope of the inductive search algorithm to a finite set of relations[1] among the search space members. If relations are important to consider, then we should pay careful attention to determine which relation is 'appropriate' and which is not.

Let us illustrate the idea. Suppose we would like to identify the person among several in a room with the largest amount of money in his or her pocket. To do better than enumeration, i.e. exhaustively picking every person and checking the amount of money they have, we must make intelligent guesses by observing certain features of the people (e.g. quality of dress, shoes etc.) If we consider 'all possible features' we are back to enumeration (Watanabe 1969; Mitchell 1980). We must consider a certain finite set of features that defines the

[1]A relation is defined as a set of ordered tuples. A class is a tuple of elements taken from the domain under consideration. In this document we will primarily be concerned with tuples taken from space of n-ary Cartesian products of the search domain with itself. Equivalence relations are symmetric, transitive, and associative relations; similarity based equivalence relations among a space of binary sequences define equivalence based on similarity among the sequences.

bias of the process. Features like quality of dress define relations among the set of people. Depending on what we mean by 'quality of dress', such a relation may divide the set of people into different classes, such as cheaply dressed people, very expensively dressed people, and so on. We consider hypotheses defined by the feature set, use it to divide the search space into different classes, and evaluate hypotheses using samples taken from the search domain. The set of features, that we restrict our attention to, may be pre-determined or dynamically constructed during the course of induction. The decomposition of BBO in SEARCH in terms of relation, class, and sample spaces essentially captures this idea. Two important underlying processes of a BBO algorithm are, (1) *construction of partial ordering, followed by selection among relations* and (2) *construction of partial ordering, followed by selection among classes*. Note that the former step is essential since some relations are inherently good and some are not. For example, 'quality of dress' may be a good relation, but 'colour of hair' may not. In SEARCH, relations that are inherently good for decision making are said to *properly delineate* the search space. If we construct a partial ordering among the classes, defined by a relation of order k (logarithm of the number of classes defined by the relation), select the 'top' ranked classes for further exploration, and the class containing the optimal solution is one among those selected classes, then we say that order-k relation *properly delineates* the search space. The search for appropriate relations and classes can be viewed as decision making processes in the relation and class spaces respectively. SEARCH offers a general probabilistic and approximate framework to do that. If the relation space provided *a priori* to the search algorithm contains all the relations needed to solve a problem and the order of all of these suitable relations is bounded from top by some constant k, then the given problem can be solved in sample complexity (can be loosely defined as the number of samples taken for solving the problem) polynomial in problem size, solution quality, success probability. This class of problem is called the class of order-k delineable problems.

SEARCH points out that, since induction is an essential part of BBO, search for appropriate relations is critical. Instead of looking for better solutions from the beginning, SEARCH advocates a BBO algorithm to

1. detect the structure of the search space, induce relations and classes to capture that
2. identify desired quality solutions by guiding the search following the detected structure

A detailed description of each of these processes can be found elsewhere (Kargupta 1995). In order to fully appreciate the critical role of efficient relation and class searches we must understand their computational cost; the following section considers that.

4. Cost of relation and class search

For a given relation space, and a well-defined algorithm in SEARCH, it is possible to derive a bound on the number of samples (sample complexity)

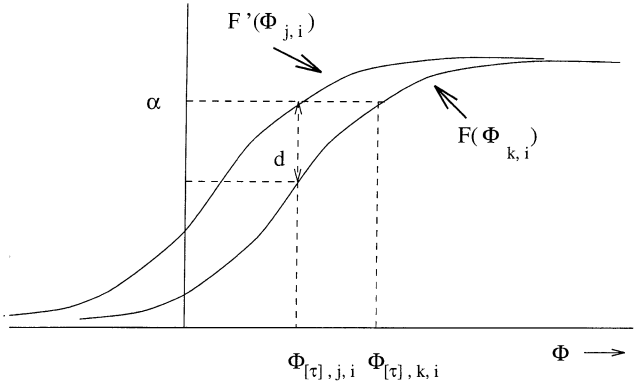

Fig. 3.2 Objective (Fitness) function value (Φ), distribution function (*F*, *F'*) of two classes $C_{j,i}$ and $C_{k,i}$.

needed for the desired solution quality and reliability of decision making. Defining an algorithm in SEARCH first requires specifying class and relation comparison statistics. Although, most of the existing BBO algorithms do not explicitly define them, SEARCH does so in order to quantify and understand the role of decision making in the relation and class spaces. Kargupta (1995) considered a distribution free ordinal comparison statistics for comparing both relations and classes. In an ordinal comparison statistic, two distributions are compared on the basis of some chosen percentile. SEARCH does not assume any particular technique for evaluating a relation or a class since typically it differs from algorithm to algorithm. It assumes that the algorithm has a way to rank the relations and classes. Figure 3.2 shows the cumulative distribution functions (cdf) F' and F of two arbitrary subsets $C_{j,i}$ and $C_{k,i}$, respectively. Indices j, k represent the two classes defined by some relation r_i. When these two classes are compared on the basis of the α quantile, then we say $C_{j,i} \leqslant \alpha C_{k,i}$, since $\Phi_{[\tau],j,i} \leqslant \Phi_{[\tau]k,i}$; $\Phi_{[\tau],j,i}$ and $\Phi_{[\tau]k}$ are the solutions of $F'(\Phi_{j,i}) = \alpha$ and $F(\Phi_{k,i}) = \alpha$, respectively. Let us define

$$d = F(\Phi_{[\tau],k,i}) - F(\Phi_{[\tau],j,i})$$

The variables d defines the *zone of indifference*, which is basically the difference in the percentile value of $\Phi_{[\tau]k,i}$ and that of $\Phi_{[\tau],j,i}$ computed from the same cdf F. Figure 3.2 explains this definition. We can quantify the decision making process using such ordinal class and relation comparison statistics. Let Ψr be the given relation space and $S_r \subseteq \Psi_r$ be the set of relations needed to solve the given problem. We denote the index of a relation r_i by N_i. Define,

$$N_{\max} = \max\{N_i | \forall r_i \in S_r\}$$
$$d' = \min\{F(\Phi_{[\tau],*,i}) - F(\Phi_{[\tau],j,i}) | \forall_j, \forall_i\}$$

where $F(\Phi_{[\tau],*,i})$ is the cdf of the class containing the optimal solution. The index j varies over all the classes defined by a relation r_i. Index i varies over all the relations in Ψ_r. If d^* is a constant such that $d' \geq d^*$, that corresponds

to the desired quality of decision making in the class space, the bound on overall sample complexity is,

$$SC \leq \frac{N_{max} \| S_r \| \log \left(1 - \left(\frac{q}{q_r} \right)^{\frac{1}{\| S_r \| (N_{max} - M_{min})}} \right)}{-d^*} \quad (1)$$

where q is the overall desired success probability and q_r is the desired success probability in the relation space. M_{min} is a constant that depends on the memory used by the algorithm.

As we increase the success probability in the relation space, the overall success probability in the combined relation and class spaces increases. The sample complexity should therefore decrease as success probability in the relation space increases. This also shows that SC decreases with increase in q_r. Note that the ratio $\left(\frac{q}{q_r} \right)^{\frac{1}{\| S_r \| (N_{max} - M_{min})}}$ approaches one in the limit as $\| S_r \| (N_{max} - M_{min})$ approaches infinity. Therefore, SC grows at most linearly with the maximum index value N_{max} and the cardinality of the set S_r. Recall that d^* defines the desired region of indifference; in other words, it defines a region in terms of percentile within which any solution will be acceptable. The sample complexity decreases as d^* increases. Kargupta (1995) also showed that when no relations are considered, this expression points out that the sample complexity will be of the order of the size of search space; in other words search will be no better than enumeration. For a given relation space and a class of problems that can be solved considering a bounded number of relations from that space, inequality 1 gives the bound on sample complexity for desired quality and reliability of the decision making. Although different algorithms may use different class and relation comparison statistics, the overall physical implications of the different terms of the bound provide us with fairly universal insights into any BBO process. This bound can also be used to identify the class of BBO problems that we can efficiently solve. The following section presents that.

5. Difficult and easy BBO problems

Traditionally problem difficulty in the BBO is characterized by different features of the problem, such as local optima, decomposability, noise in objective function evaluation, and others. Since the success of a BBO algorithm is significantly controlled by the searches in the relation and class spaces, the notion of easy and difficult BBO problems must depend on the cost of searching in these spaces. In fact, the bound on the sample complexity, noted in the previous section, can be directly used to characterized hard and easy problems from an algorithm perspective.

Recall from inequality 1 that the sample complexity grows linearly with the cardinality of the set of relations considered to solve the problem, S_r.

Since the purpose of the relations is to relate different search space members, most of the commonly used relation spaces are of sizes exponential in the *size* of the problem. In this document, we are going to define *size* of the problem (l) as the logarithm of the cardinality of the complete search space. Typically this corresponds to the total number of dimensions of the problem. In a sequence representation with constant alphabet size, the length of the sequences needed to represent the search space may be an example of such a *size* parameter.

Definition 1 (Problem difficulty in SEARCH) *Given an optimization function* $\Phi : X \rightarrow \Re$ *(where X is finite discrete space) and a set of relations* Ψr, *we call a problem difficult for an algorithm if the total number of samples needed to find the globally optimal solution grows exponentially with l,* q, q_r $1/d^*$, *and* $1/d_r^*$.

Where q denotes the bound in the overall decision success probability in choosing the right classes; $1/d^*$ defines the quality of the desired solution. Both q and $1/d^*$ together can be viewed as representing the overall accuracy and the quality of the solution found; q_r is the bound in success probability in choosing the right relations, and $1/d_r^*$ represents the desired quality of the relations.

The above definition of problem difficulty in SEARCH can be physically interpreted into the following items.

1. Growth of the search space along problem dimension.
2. Inadequate source of relations and decision making in relation space.
3. Inaccurate decision making in choosing classes.
4. Quality of the desired solution and relations.

This gives a general description of the SEARCH perspective of problem difficulty. Since, the bound on sample complexity grows linearly with S_r, we can only handle a polynomially bounded relation space size. Since inequality 1 also grows linearly with $N_{\max}$, we also need to bound that by a polynomial of l. As noted earlier, for order-k relations $N_{\max}$ is bounded by a constant.

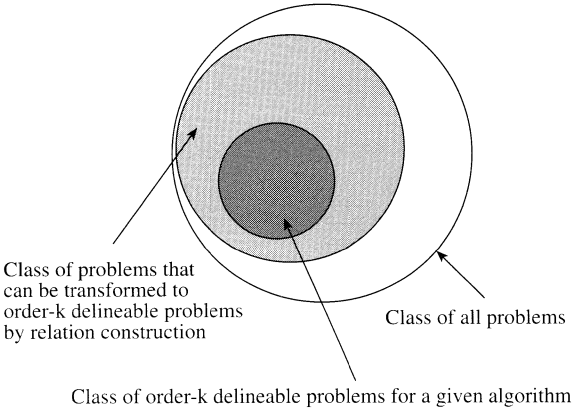

Fig. 3.3 BBO problems from the delineability perspective.

Therefore, we can solve the class of problems for which, (1) the cardinality of the set of all required delineable relations can be bound by a polynomial in l and (2) the order of each of these relations is at most k, using a polynomial number of samples. This class of problems is called the class of *order-k-delineable problems*. A formal definition of this class of problems can be found in (Kargupta 1995). Figure 3.3 depicts a conceptual hierarchy of different class of problems offered by the SEARCH framework. In the following part of this section, we shall make our notion of order-k delineable problems more tangible by considering a class of problems in the sequence space.

Let us consider sequence representation in which the underlying search space is represented by a sequence of l characters, where each character can take a value from the alphabet set, Λ. The cardinality of Λ will be denoted by λ. Let us also restrict our attention to similarity based equivalence relations and classes. In the GA literature they are usually called partitions and schemata respectively. The overall relation space, $\Psi_r = \{f, \#\}^l$, where f indicates values that must match for equivalence and # is a wild character that matches any value. The cardinality of the set of all such similarity based equivalence relations $\| \Psi_r \| = 2^l$. Since this grows exponentially with l, we cannot afford to consider all such similarity based relations. One way to pick up a generally meaningful polynomially bounded subset of Ψ_r is to consider only those relations in which there are at most k fixed position of equivalence (f) and $l - k$ wild characters. There are $\binom{l}{k}$ such similarity based equivalence relations and it is a polynomial of l for constant values of k. Since GAs often deal with sequence representation and similarity based equivalence relations, such a restriction must be imposed in order to guarantee polynomial time search. Therefore, this class of order-k delineable problems should be of primary importance in the context GAs. Even if we have interest in other paradigms of evolutionary computation that do not pay attention to similarity based equivalence relations as such, the SEARCH framework points out that,

1. careful attention must be paid to the relation induction process and
2. order-k delineable problems must be identified for the chosen relation space of the algorithm.

In this chapter, we focus on GAs and therefore we restrict our attention to the class of order-k delineable problems in the sequence space. Although the relation space size is bounded by a polynomial, efficient detection of relations and classes by choosing appropriate statistics is a major challenge. Like any other BBO algorithm, the GAs need to pay careful attention to the relation induction process for solving the order-k delineable problems efficiently. In this chapter we are therefore interested in developing scalable GAs that can solve order-k delineable problems efficiently. First, let us investigate the efficacy of simple GAs in accomplishing this task.

6. The simple genetic algorithms

The simple Genetic Algorithm (sGA) (Dejong 1975; Goldberg 1989; Holland 1975) is a popular class of genetic algorithms. The simple GA uses operators

like selection, crossover, and mutation to explore the search space adaptively in order to maximize or minimize the objective function (sometimes called *fitness* function). Simple GA typically uses a sequence representation. In other words, the search variables are represented as a sequence (often called *chromosome*) of symbols, chosen from some given alphabet set. The slot corresponding to any entry in the sequence is called a *gene*. Popular approaches include binary, gray, and real value codings of the search variables. A simple GA starts from a randomly generated population. It iteratively applies the search operators — *selection, crossover*, and *mutation* — to this population to produce a new population of chromosomes. The main search operators are as follows.

1. Selection: Compute the objective (fitness) function values of all the chromosomes. Make more copies of the chromosomes with higher fitness and use these additional copies to replace those chromosomes of the population that have worse objective function values.
2. Crossover: The crossover operator is usually applied on the population with a high probability. There are several types of crossover operators prevailing in the GA literature. A simple one-point crossover picks two chromosomes from the population randomly. Next it picks a random cross-point (i.e. a slot in the chromosome) that divides each chromosome into two halves. This is followed by swapping of either the left or right halves.
3. Mutation: Simple mutation is usually a low profile operator that changes the value of a gene with some low probability.

Although the sGA explicitly processes a population of chromosomes, a better understanding about the underlying search may be obtained by investigating the processing of *schemata* (similarity based equivalence classes) in the population (Holland 1975). In a sequence representation, a similarity based partition divides the space of all sequences into different schemata. For example in a 4-bit representation schema (singular form of schemata) 11## denotes the set of all strings that start with a 11 (i.e. the set {1100, 1101, 1110, 1111}). The corresponding partition can be represented by *ff*##, where *f* denotes the position of equivalence and # denotes the wild card character. Partition *ff*## divides all strings into four schemata namely 00##, 01##, 10## and 11##. The effect of selection, crossover, and mutation applied on the population can also be interpreted in the space of partitions and schemata. For a given population of strings and GA operators the so called *schema theorem* (Holland 1975) can be used to determine an expected bound on the growth or destruction of schemata from generation to generation.

The simple GA has been quite successful in solving many different problems (Goldberg 1989; Mitchell 1996); however it is not magic. Like any other BBO algorithm, the sGA is fundamentally based on an inductive search process.

Therefore, the observations made by the SEARCH framework are equally applicable to sGA. The success of GAs depends on several factors, including:

1. detection of schemata that capture the desired solution(s);
2. interaction between schemata and genetic operators;
3. population size;
4. problem difficulties.

Although schemata and partitions are often used as a tool to understand the underlying behaviour of GAs, the sGA does not have any mechanism for explicit processing of schemata and partitions. This implicit perspective alone falls far short in delivering reliable performance for schema and partition detection. A simple illustration can be given using the following example. Consider the deceptive trap (Ackley 1987) function, $f(x) = k$ if $u = k$; $f(x) = k - 1 - u$ otherwise; where u is the unitation variable, or the number of 1s in the string x, and k is the length of the sub-function. This function is widely reported to be difficult for simple GA since low order partitions lead sGA toward the wrong direction. If we carefully observe this trap function, we shall note that it has two peaks. One of them corresponds to the string with all 1s and the other is the string with all 0s. The solution with all 1s is optimal. Let us construct a test function by concatenating 5-bit trap functions one after another. In other words each consecutive five bits define a separate trap function. This overall concatenated function can be linearly decomposed into order-5 sub-functions. To solve this problem efficiently either the sGA needs to be informed about related bits or they must be adaptively detected by selecting the appropriate partitions and schemata. Lack of a mechanism for explicit partition and schema detection makes the sGA perform very poorly for such problems. Figure 3.4 shows the result of a typical sGA run for a 36-bit objective function, comprised of six trap sub-functions, each of size six bits. The sGA fail to obtain the optimal solution.

Many real life problems exist in which the underlying non-linearity of the problem results in higher order problem delineability. As a result, success

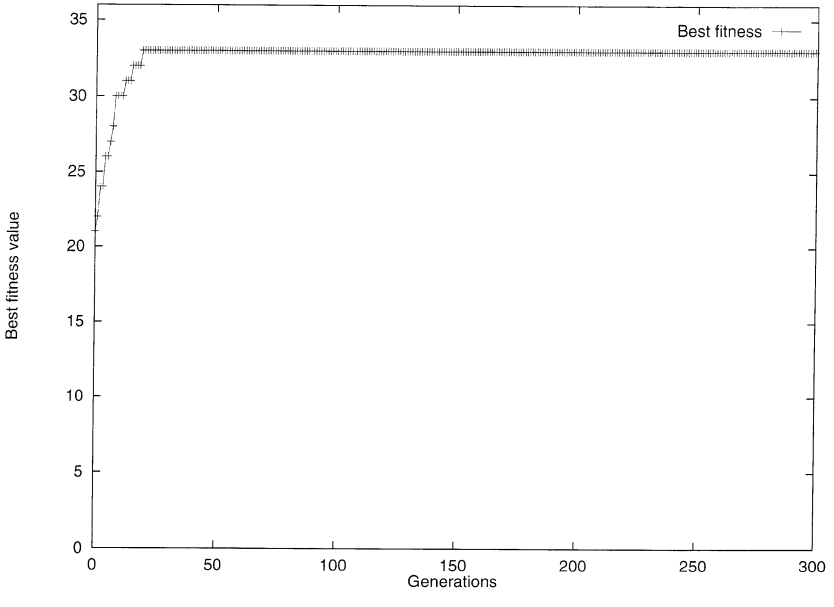

Fig. 3.4 The variation of the best fitness value of an sGA population with respect to different generations. The optimal solution has a fitness value of 36. The sGA has population size = 100, crossover probability = 0.7, mutation probability = 0.001, binary tournament selection.

demands effective search for appropriate partitions and schemata. This problem of detecting appropriate relations and classes is traditionally called linkage learning in the GA literature. Although by definition linkage learning is not necessarily restricted to similarity based relations and classes, in the following part of this chapter linkage learning will be restricted to only those special cases. Linkage learning is essentially the problem of detecting appropriate bases of the underlying problem representation. A general definition of linkage learning can be found elsewhere (Kargupta 1998; Kargupta and Bandyopadhyay, to be published). There is a growing consensus that scalable linkage learning is essential for the success of GAs in search, machine learning, optimization, and data mining problems. However, the need for linkage learning was realized even during the early inception of the GAs. The following section describes the related work since the dawn of genetic algorithm research.

7. Linkage learning in simple GAs

The efficacy of implicit processing has been questioned since the inception of the GAs. Several efforts have been made for designing GAs capable of explicit detection of significant partitions and schemata. The history of linkage learning efforts dates back to Bagley's dissertion (Bagley 1967). Bagley uses a representation in which the gene explicitly contains both the position and the allele value. For example, string ((0 1)(2 0)(1 1)) will correspond to the string 110 in a fixed-locus representation of the simple GA. Bagley used the so-called *inversion* operator for adaptively clustering the related genes that define good partitions and schemata. The inversion operator works by reversing the order of the genes lying in between a pair of randomly chosen points along the chromosome. Although this mechanism was used for generating new tightly coded partitions, Bagley's work provides no mechanism for accurate evaluation of the partitions. Moreover, introduction of the inversion operator restricted the use of GA crossover operator and Bagley did not conclude in favour of the use of inversion. Rosenberg (Rosenberg 1967) also investigated the possibility of learning linkage by evolving the probability of choosing a location for crossover. Although this approach does not rigorously search for appropriate partitions, adaptive crossover point may be able to process schemata, with widely separated fixed bits, better than a single point crossover. Frantz (Frantz 1972) investigated the utility of the inversion operator and like Rozenberg reported that inversion is too slow and not very effective. Holland (Holland 1975) also realized the role of linkage learning and suggested the use of the inversion operator despite its reported failure in earlier studies. Goldberg and Lingle (Goldberg and Lingle 1985) introduced a new PMX crossover operator that could combine the ordering information of the selected regions of the parent chromosomes. They concluded that this approach has more potential than the earlier approaches. Schaffer and Morishima (Schaffer and Morishima 1987) introduced a set of flags in the representation. These flags were used for identifying the set of genes to be used for crossover points. For different test

problems, they noted the formation of certain favourite crossover points in the population, that corroborated their hypothesis regarding the need for detecting gene linkage. Goldberg and Bridges (Goldberg and Bridges 1990) confirmed that lack of linkage knowledge can lead to failure of GAs for difficult classes of problems, such as deceptive problems. Additional efforts on linkage learning GAs can be found elsewhere (Levenick 1991; Paredis 1995; Smith and Fogarty 1996). Harik introduced the LLGA (Harick 1997) which made an effort to learn linkage by introducing the so-called *exchange crossover operator* and the *probabilistic expression* based representation. An alternate approach to linkage learning can be found elsewhere (Smith and Fogarty 1996).

In addition to growing empirical evidence for the need of explicit linkage learning algorithms in the GAs, theoretical advances have also started corroborating these observations. Efficacy of such implicit processing of relations has been seriously questioned on theoretical grounds (Goldberg, Korb and Deb 1989; Kargupta 1995; Thierens and Goldberg 1993). Thierens and Goldberg showed that simple GA fails to scale up for the class of problems with only order-k significant partitions, unless information about the appropriate partitions is provided by the user.

The main reasons behind this lack of scalability are the merger of the relation, class, and sample spaces into a single population and the lack of adequate efforts to methodically search for the appropriate order-k partitions. The sGA also has some additional problems in the context of efficient partition search. A single sample from the search space can be used for the evaluation of all the relations under consideration. This is because that sample must belong to some schema defined by any partition. This is often called implicit parallelism in the GA literature. Although this can be exploited in a very systematic manner when relations are methodically processed, implicit processing of schemata makes this quite noisy in the sGA. These observations regarding the problems of simple GA in searching appropriate partitions and schemata resulted in the development of different evolutionary algorithms that pay attention to the linkage learning issue. The following section describes one such algorithm, called the gene expression messy genetic algorithm.

8. The gene expression messy GA

The Gene Expression Messy Genetic Algorithm (GEMGA) (Kargupta 1996*a*, *b*; Bandyopadhyay, Kargupta, and Wang 1998) offers an efficient approach to learning linkage. In GEMGA, the problem of associating similarities among chromosomes with similarities of their corresponding fitness values is posed as a problem of detecting approximate symmetry. Symmetry can be defined as an invariance in the pattern under observation when some transformation is applied to it. Similarities among the fitness values of the members of a schema can be viewed as a kind of approximate symmetry, that remains invariant under any transformation that satisfies the closure property of the schema. The GEMGA identifies the fitness symmetry (in an approximate sense) by identifying symmetry breaking dimensions, and uses them to detect

the underlying linkage. This approach appears to be quite successful for properly delineating partitions of an optimization problem. An additional strength of this method is its intelligent crossover, which explicitly works to preserve linkage information detected earlier in the algorithm. Unlike other approaches (Deb, Harik etc), which relied on a position independent coding and tried to get a *'tight linkage'*, this approach makes no such attempt; rather, it starts by detecting local fitness symmetry and uses that to do a bottom-up construction of global linkage information. In the following sections we review the recently proposed version (v.1.3) (Bandyopadhyay, Kargupta, and Wang 1998) of the GEMGA.

8.1 Population sizing

In order to detect a schema, the GEMGA requires that the population contains at least one instance of that schema. If we consider the population size to be a constant and randomly initialized, then this can be guaranteed only when the order (number of fixed positions) of the schema to be detected is bounded by some constant k. For a sequence representation with alphabet set Λ of cardinality λ, a randomly generated population of size λ^k is expected to contain one instance of every order-k schema. The population size in GEMGA is therefore $m = c\ \lambda^k$, where c is a constant. Although we treat c as a constant, c is likely to depend on the variation of fitness values of the members of the schema. Note that the population size is independent of the problem size l. For all the experiments reported in this work, the population size is kept constant.

8.2 Representation

The GEMGA uses a sequence representation. Every member of this sequence is called a *gene*. A gene is a data structure, which contains the *value* and *capacity*. The *value* contains the value of the gene, which could be any member of the alphabet set, Λ. The *capacity* associated with every gene takes a positive real value. The chromosome also contains a dynamic list of lists called the *linkage set*. The *linkage set* of a chromosome is a list of weighted lists. Each member of this sequence consists of a list, termed *locuslist* which defines the set of genes that are related, and three factors, the *weight, goodness*, and *trials*. The weight is a measure of the number of times that the genes in locuslist are found to be related in the population. The goodness value indicates how good the linkage of the genes is in terms of its contribution to the fitness. This value is normalized between 0 and 1, and is initialised to 0. The trial field indicates the number of times this linkage set has been tried. (Note that if the trial of any element of the linkage set is zero, then its goodness is temporarily assumed to be 1, unless proved otherwise.) The *linkage set* space over all genes defines the relation space of the GEMGA. Figure 3.5 shows the structure of a chromosome in GEMGA. Unlike the original messy GA (Deb 1991; Goldberg, Korb and Deb 1989) no under or over-specifications are allowed. A population in GEMGA is a collection of such chromosomes.

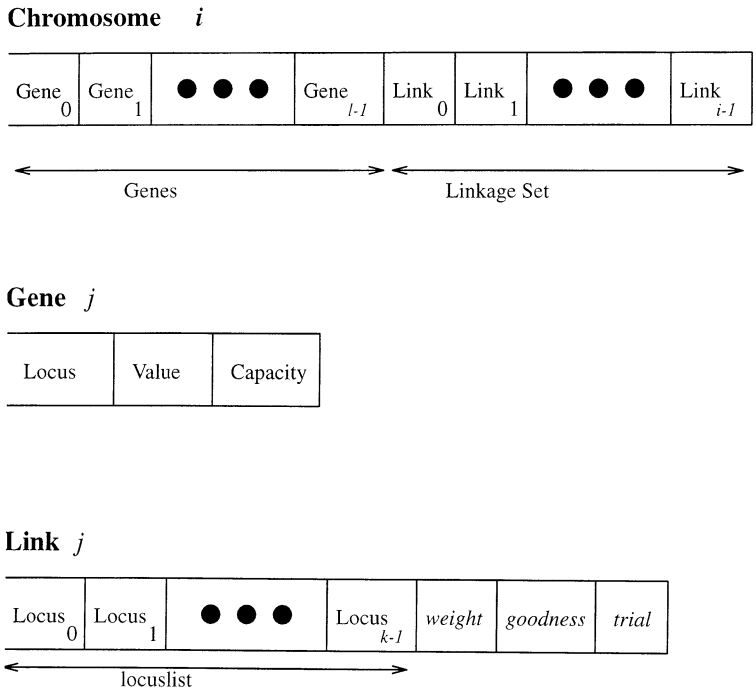

Fig. 3.5 Structure of a chromosome in GEMGA.

8.3 Operators

The GEMGA has three primary operators, namely: (1) *Transcription*, (2) *PreRecombinationExpression* and (3) *RecombinationExpression*. Each of them is briefly described in the following.

8.3.1 Transcription

The GEMGA *Transcription* operator plays an important role in the detection of schemata. It detects *local* symmetry in the fitness landscape by noting the relative invariance of the fitness values of chromosomes under transformations that perturb the chromosome, one gene at a time. It changes the current value of a gene to a different value, randomly chosen from the alphabet set and notes the change in fitness value. If the fitness deteriorates because of the change in gene value, that gene is identified as the symmetry breaking dimension and the corresponding gene capacity is set to zero, indicating that the value at that gene value cannot be changed. On the other hand, if the fitness improves or does not change at all, the corresponding capacity of the gene is set to one, indicating that this dimension offers symmetry, with respect to the pattern of improvement in fitness. Finally, the value of that gene is set to the original fitness. This process continues for all the genes and finally all the genes whose capacities are reduced to zeroes are collected in one set, called the initial locuslist. This is stored as the first element of the linkage set associated with the chromosome. Its weight, goodness, and trial factors are initialised to 1, 0, and 0 respectively. The transcription operator does not change

```
1  def Transcription (self):
2    for a ∈ self.pop:
3      tempset ← kjSet()
4      value ← a.value
5      for j ∈ range ( self.problem_length ):
6        a.flip (j)
7        if value < objective (a):
8          a.genes [j].weight ← 0.0
9          tempest.add(j)
10       else:
11         a.genes [j].weight ← 1.0
12       a.flip(j)
13   a.linkageset.append( link (tempest, self.INITIAL_WEIGHT,0,0))
14 return
```

Fig. 3.6 Transcription operator for minimization problem.

anything in a chromosome except the capacities and it initiates the formation of the linkage sets. Any symmetry that remains true over the whole search space also remains true within a local domain. Figure 3.6 shows the Transcription operator.

Locally detected schemata are next evaluated in a population-wide global sense, as described in the following section.

8.3.2 PreRecombinationExpression

The *PreRecombinationExpression* stage detects schemata that capture symmetry beyond a small local neighbourhood defined by the bit-wise perturbation of transcription. The PreRecombinationExpression phase determines the clusters of genes precisely defining the relations among those instances of genes. It consists of two steps ResolveLinkage, and GetFinalLinkage.

```
1  def PreRecombinationExpression( self ):
2    for a ∈ self.pop:
3      listx ← range ( len (self.pop ))
4      for b ∈ range (self.no_linkage.exp ):
5        while 1:
6          chosen ← whrandom.choice (listx)
7          listx.remove ( chosen )
8          ch ← self.pop [ chosen ]
9          if ¬ ( ch = a ):
10           break
11       self.ResolveLinkage ( a, ch )
12     counter ← counter + 1
13     print counter
14   for a ∈ self.pop:
15     a.GetFinalLinkage ( self.EPSILON, self.WEIGHT_THRESH )
16   return
```

Fig. 3.7 PreRecombinationExpression Operator.

ResolveLinkage Each chromosome in the population undergoes a fixed number (No_Linkage_Exp) of ResolveLinkage operations with different members of the population other than itself. During the ResolveLinkage operation those genes which are members of the initial locuslist (constructed using Transcription) of both the chromosomes and having the same value and capacity are grouped together in a new set. If the set is already present in the linkage set of the first chromosome then the weight of the corresponding locuslist is increased by an amount INCR_WEIGHT, else it is included as a new element of the linkageset.

GetFinalLinkage A conditional probability matrix P is now constructed, where the entry $P[i,j](i \neq j)$ denotes the probability of finding gene i in a locuslist given gene j is already present. In case $(i = j)$ it denotes the probability of a locuslist containing gene i only. The maximum probability *max[i]* in each row is calculated and those entries which are less than *max [i] – EPSILON* are replaced with 0. Now a new linkage set is calculated by collecting the nonzero entries in each row into a locuslist and using the mean of the corresponding probabilities as its weight. The set is added to the new linkage set if its weight is greater than WEIGHT_THRESH. The addition of a new locuslist is done in the same way as was done during the resolution phase by checking for the existence of another locuslist with the same members as the new one.

```
1  def GetFinalLinkage ( self, EPSILON, WEIGHT_THRESH ):
2    toplist ← self.linkageset [0].set.items()
3    conditional ← self.CalcConditional()
4    ThresholdConditional ( conditional, EPSILON )
5    self.linkageset ← [ ]
6    for i ∈ toplist:
7      sum ← 0.0
8      count ← 0.0
9      newlink ← kjSet( )
10     for j ∈ toplist:
11       if i = j:
12         newlink.add(j)
13       else:
14         if conditional [I][j] ≥ max [I]:
15           newlink.add(j)
16           sum ← sum + conditional [I] [j]
17           count ← count + 1
18     elem ← len ( newlink )
19     if count:
20     sum ← sum / count
21     if elem Λ ( sum > WEIGHT_THRESH ):
22     self.linkageset.append(link ( newlink, sum, 0.0, 0.0 ))
23   return
```

Fig. 3.8 GetFinalLinkage Operator.

```
1  def RecombinationExpression ( self ):
2    newpop ← [ ]
3    for i ∈ range ( len (self.pop )):
4      maxind ← i
5      maxfit ← self.pop [i].value
6      for j ∈ range ( 1, self.tsixe ):
7        tempind ← whrandom.randint ( 0, len ( self.pop ) - 1 )
8        if maxfit > self.pop [ tempind ].value:
9          maxind ← tempind
10           maxfit ← self.pop [ tempind ].value
11     newpop.append ( self.pop [ maxind ] )
12   self.pop ← newpop
13   for i ∈ range ( 0, len ( self.pop ), 2 ):
14     self.GEMGARecombination ( self.pop [i], self.pop [i + 1])
15   return
```

Fig. 3.9 RecombinationExpression Operator.

8.3.3 RecombinationExpression

The *RecombinationExpression* Phase is the selecto-recombinative phase of the algorithm and is run a fixed (No_Gen) times. It also consists of two steps. First a mating pool is created by performing binary *tournament selection* in the population. Then the GEMGA *Recombination* operator is applied iteratively over pairs of chromosomes.

First, copies of the given pair are made, and one of them is marked. An element of the linkage set of the marked chromosome is selected, based on a linearly combined factor of its weight and goodness, for swapping. The corresponding genes were swapped between the two chromosomes provided the goodness values of the disrupted locuslists of the unmarked chromosome are less than that of the selected one. The linkage sets of the two chromosomes are adjusted accordingly. Depending on whether the fitness of the unmarked chromosome decreases or not, the goodness of the selected linkage set element is decreased or increased. Finally, only two of the four chromosomes (including the two original copies) are retained (Bandyopadhyay, Kargupta, and Wang 1998).

8.4 The algorithm

The overall structure of the GEMGA is summarized below:

1. Initialization Randomly initialize the population.
2. PrimordialExpression Detect schemata that capture local fitness symmetry by the so called *transcription* operator. Since population size $m = c\lambda^k$, this can be done in time $O(\lambda^k l)$.
3. PreRecombinationExpression Identify schemata that capture fitness symmetry over a larger domain. This only requires comparing the chromosomes with each other and no additional function evaluation is needed.

4. RecombinationExpression

(a) Create a Mating Pool using Binary Tournament Selection.

(b) GEMGA recombination: The GEMGA uses a recombination operator, designed using motivation from cell meiosis process that combines the effect of selection and crossover. Reconstruct, modify schema linkage sets and their parameters.

(c) Mutation: Low probability mutation like simple GA. All the experiments reported in this work used a zero mutation probability.

The primordial expression stage requires $O(\lambda^k l)$ objective function evaluations. PreRecombinationExpression requires $O(\lambda^{2k})$ pair-wise similarity computation time (no objective function evaluation). The length of the Recombination stage can be roughly estimated as follows. If t be the total number of generations in the juxtapositional stage and if selection gives α copies to the best chromosome then if selection dominates the crossover, every chromosome of the population will converge to the same instance when $\alpha^t = m$,

$$t = \log m / \log a \tag{2}$$

Substituting $m = c\lambda^k$, we get:

$$t = \frac{\log c + k \log \lambda}{\log a} n \tag{3}$$

Therefore, the number of generations in the recombination expression stage is $O(k)$. This result is true when selection is allowed to give an exponentially increasing number of copies. The overall number of function evaluations is bounded by $O(\lambda^k l)$. This analysis assumes that the cardinality of the alphabet set of the chosen representation is bounded by a small constant (e.g. in case of binary representation it is two). The GEMGA has been applied to solve a feature selection problem using a power distribution network fault-diagnosis data set. The following section describes the application domain and presents the results.

9. Data analysis application

This section describes the development of a diagnostic tool for fault identification and location in a radial medium voltage (MV) distribution network using the GEMGA algorithm. A minimal subset from a large set of electrical features available at the metering stations is identified. Finding the most significant patterns or parameters in a large database is not a trivial problem; solving it however can help speed up on-line diagnosis of faults. The evaluation of the significance of each feature is done by means of a diagnostic model based on the comparisons between the network simulator output in normal and outage operating states. The characterization of each faulty condition is extensively studied in (Di Silvestre 1998). The results

```
1  def GEMGARecombination(self, chrom1, chrom2);
2    gl ← []
3    tl ← []
4    swapflag ← 0
5    downflag ← 0
6    equalflag ← 0
7    t1 ← copy.deepcopy(chrom1)
8    t2 ← copy.deepcopy(chrom2)
9    if chrom1.linkageset:
10     maxind ← chrom1.ChooseLinkage()
11     maxtrial, maxgood, maxwt1 ←
chrom2.MaxDisruptedLinkage(chrom1.linkageset[maxind])
12       if chrom1.linkageset[maxind].goodness ⩾ maxgood:
13         chrom1.swapgenes(maxind, chrom2)
14         chrom1.fitness()
15         chrom2.fitness()
16         tl ← chrom2.AdjustLinkage(chrom1.linkageset[maxind])
17         index ← len(chrom2.linkageset)-1
18         chrom1.SelfAdjustLinkage(chrom2.linkageset[-1],tl)
19         chrom2.linkageset ← chrom2.linkageset + gl
20         if chrom2.value ⩽ t2.value:
21           chrom2.IncreaseGoodness()
22           if chrom2.value = t2.value:
23             equalflag ← 1
24           elif chrom2.value > t2.value:
25             chrom2.DecreaseGoodness()
26             downflag ← 1
27           if (chrom2.linkageset[index].goodness ⩾
maxgood) Λ (maxtrial > 0) Λ (chrom1.value > t1.value):
28             self ← t1
29           if equalflag:
30             self ← t1
31             chrom2 ← t2
32           if downflag:
33             chrom2 ← t2
34     return
```

Fig. 3.10 GEMGARecombination Operator.

presented here provide a considerable number of different possible minimal sets of features and the results of the algorithm can be interpreted in physical terms.

9.1 Background

The problem of fault identification and location in power distribution systems has created wide interest among the scientific community in the Power

```
1  def main()
2    GA ← GEMGA()
3    GA.Randomize()
4    GA.Transcription()
5    GA.PreRecombinationExpression()
6    gen.counter ← 100
7    for j ∈ range (gen_counter):
8      GA.RecombinationExpression()
9      GA.PrintBest(j)
10   return
```

Fig. 3.11 Main loop of the algorithm.

Distribution area. An efficient approach to this problem leads to immediate benefit to the customer in service quality and cost reduction. Electrical utility companies are therefore interested in efficient tools for fast and accurate diagnosis of the system so that down-time can be minimized. In general diagnostic problems can be divided in two sub-problems:

1. building up a reliable knowledge base for fault-identification;
2. identification of faults.

The problem of identifying the significant features required to perform a fast online diagnosis can be done either on-line or off-line. Examples of the first approach can be found in (Wen and Chang 1997) and (Chang *et al.* 1997). In Wen and Chang (1997), the identification of the faulty sub-network is performed using the real-time network topology determination method and by comparing the pre-fault and post-fault conditions. In this way, the scope of the fault section estimation can be drastically reduced and the diagnostic problem is then solved using a refined GA. In Chang *et al.* (1997), the problem size is reduced on-line. This is done using a fuzzy expert system for faulty sub-network identification. The knowledge base is enriched on-line by post-fault data of protective relays and circuit breakers, collected from all over the network. The problem of detecting precise fault location is considered elsewhere (Zhu *et al.* 1997) and (Hsu *et al.* 1991). A multi-stage scheme for locating and diagnosing faults in distribution feeders has been developed by (Zhu *et al.* 1997). Fault location detection is performed using on-line data collected from the Medium Voltage/Low Voltage (MV/LV) sub-station and using probabilistic modelling and analysis. The results of the first location detection are then adjusted taking into account the approximation introduced in the generated models. In (Hsu *et al.* 1991), the problem of the fault location is solved using an expert system. Recently, the large amount of information derived from dispatchers' past experience and logical reasoning have prompted extensive work on the application of expert systems in fault diagnosis. Both rule-based and model-based expert systems can have drawbacks due to either the large number of rules needed to describe the protection system behaviour and/or due to the time required by the inference process. In (Momoh *et al.* 1997) the faults diagnosis is performed for single-line faults in

distribution systems, using on-line digitized data collected at the High Voltage/Medium Voltage (HV/MV) sub-station. It describes an integrated package for fault diagnosis, using a rule based scheme and an artificial neural network to identify, detect and classify single-phase faults. The proposed diagnostic tool can be applied to distribution networks with grounded or ungrounded neutral. Togami and Kitahashi (Togami *et al.* 1995) have developed a diagnosis methodology using a decision tree. The method is applied to a single main MV feeder with burdens at the end of the feeder. It takes into consideration only two types of faults. An alternate fault diagnosis system has been developed by Teo (Teo 1995) using a special purpose machine learning algorithm. In the current study a diagnostic strategy based on off-line determination of a set of features similar to that described on Togami *et al.* (1995) and Teo (1995) is presented. In particular, in Togami *et al.* (1995), the feature selection is performed using comparison among ranges of variation of some electrical features as the parameters vary. The test system is small and the number of events is limited. In the present study however, the network model is accurate and is capable of simulating a large number of possible electrical network working conditions, in both outage and normal operating states. Moreover, the solution sets obtained are composed of electrical features that are good for distinguishing among the various faults and identifying them.

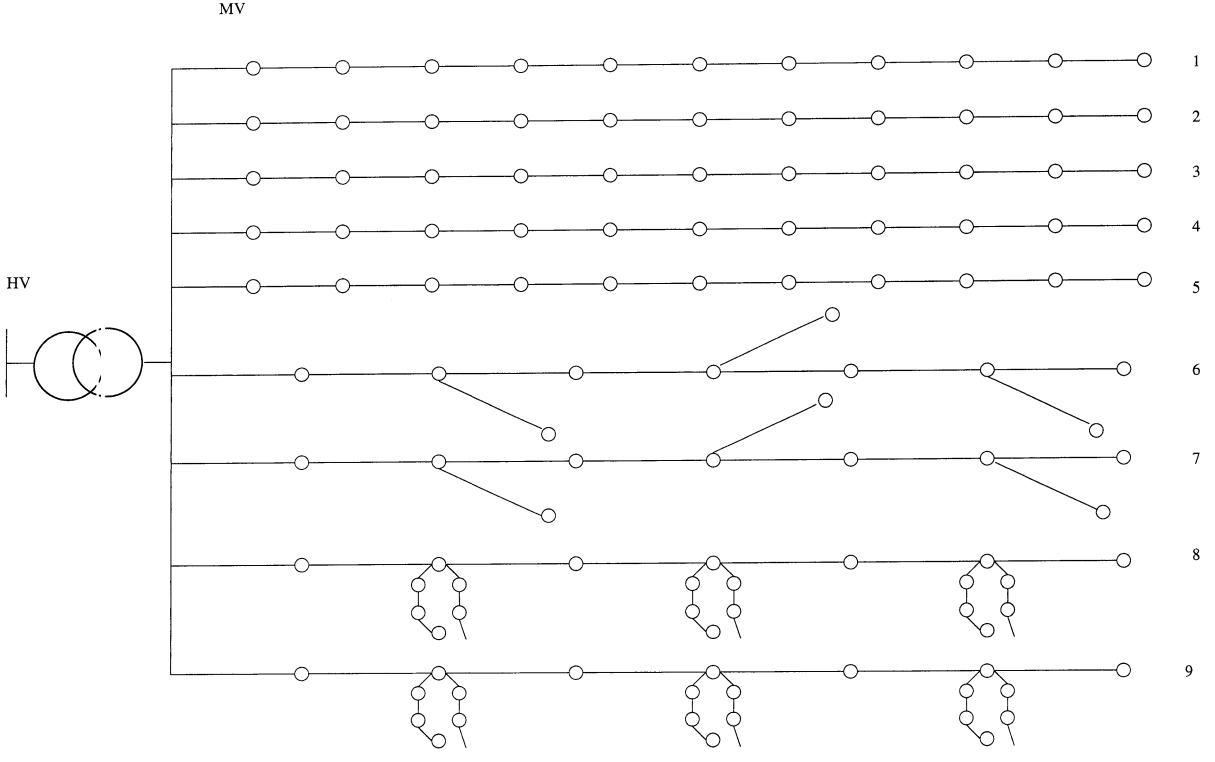

Fig. 3.12 The test system.

9.2 Problem description

The distribution network considered in this chapter is presented in Fig. 3.12. Further details regarding its electrical and topological features can be found in (Di Silvestre 1998). It has the following characteristics:

1. Single HV/MV sub-station from which a number of main feeders spread out; these supply all the radially connected loads. The connections may be cable or overhead lines.
2. Loads are supplied through MV/LV transformers installed in sub-stations, where it is possible to perform measurements of the electrical features, above and below the derivation.
3. Load varies with daily, weekly, and monthly cadence.

The faults considered here can be divided into two main categories:

1. Faults due to line insulation break-down, that can be further divided into:
   (a) single-phase faults,
   (b) phase-to-phase faults,
   (c) phase-to-phase-to-ground faults.
2. Faults due to the mechanical break-down of a line. This is primarily expected from overhead lines and can be further divided into:
   (a) direct: interruption close to a pole, when the line on the supply side goes to the ground,
   (b) inverse: interruption close to a pole, when the supply on the load side goes to the ground,
   (c) double: both sides of the lines go to the ground.

In faulty conditions the parameters that most directly influence electrical features are.

1. Neutral grounding system.
2. Fault resistance.
3. Supplied load entity at the moment the fault occurred.
4. Fault location in the network.

The electrical features considered to be useful for the diagnosis are the zero, positive, and negative components of the voltage and the current; the negative and zero components of real and reactive power. They are evaluated at the input and/or output sections of the MV/LV sub-stations; those sections will be called *control points*.

Since we assume that the system is symmetrical, choosing zero and negative real and reactive power as features allows the identification and location detection of non-symmetrical faults. The elementary *event* that can take place in the system under current consideration is one of the faulty conditions indicated above occurring between two adjacent MV/LV sub-stations. Once the parameters influencing any faulty conditions have been defined, it is possible to evaluate the corresponding range of variation of all the features at all control points, for these events. In the present application the parameters are those listed earlier. To develop a diagnostic model, the knowledge base is therefore made up of the range of variation of each of

the features considered in any of the control points. The knowledge base is made up of all the ranges defined by two real values indicating their upper and lower bounds. The simulation software for the system's behaviour is that described in (Augugliaro *et al.* 1996). The following section presents the diagnostic model used.

10. Diagnostic model

The diagnostic model comprises a system having c control points, at each of which f features can be observed, for e different types of possible events. Let us denote the minimum and the maximum value of the feature F by $F_{\min}$ and $F_{\max}$. A database of the ($F_{\min}$, $F_{\max}$) tuples is generated by varying the parameters between their respective minimum and maximum values. The total number of such tuples is therefore $c \times f \times e$. The problem can then be expressed (Di Silvestre 1998) as follows. Given the ranges of variation of all the independent parameters, identify the set of features that allows proper distinction between the various faults with a given precision. For any event i, let us denote the ranges of variation of feature F by the tuple ($F_{i,\min}$, $F_{i,\max}$). For any two such events i and j, there are three possible options for this range: non-overlapping, partially overlapping, and totally overlapping. These options are presented in Fig. 3.13.

Let us denote the range of variation of feature F, during event i by the symbol Δi. If the overlapping area S_{ij} between the events i and j is greater than zero (as in cases b) and c) shown in Fig. 3.13), then we can define index $I_{G,ij}$ as follows.

$$I_{G,ij} = 1 - \frac{S_{ij}}{\Delta_i} \tag{4}$$

This index measures the significance of F with respect to its ability to distinguish between events i and j and vice-versa. This identification index ranges

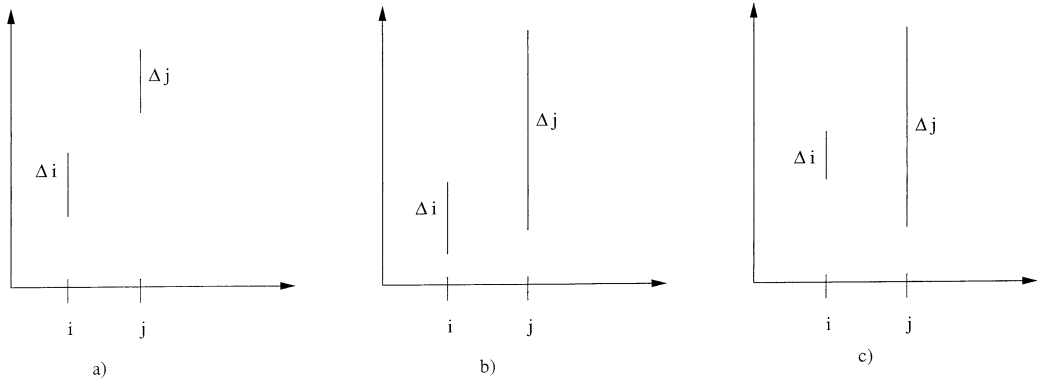

Fig. 3.13 Possible relations between the ranges of variation of a feature when events i or j take place.

between 0 and 1. Therefore, for each of the features it is possible to create a matrix whose row and column indices are the events to be recognized. This matrix provides valuable indication of the significance of a feature in distinguishing between all the events, under the assumption of uniform occurrence of all the different parameter values. In the present study, for the sake of simplicity we will consider $I_{G,ij}$ to be Boolean. Whenever $I_{G,ij}$ is lower than unity, it is set to zero. Therefore, the resulting matrices for different features are symmetrical and can be summarized into a single table with f columns and r rows. The number of rows of the table is,

$$r = \binom{e_t}{2} \tag{5}$$

where e_t is the total number of events. The problem is then to find the minimal set of features that can distinguish between all events. The test system is represented in Fig. 3.12. It comprises different types of lines such as:

1. two lines of A type, overhead lines, total length: 21 km, supplying 10 MV/LV sub-stations whose rated power is 250 kVA;
2. five lines of B type, cable lines, total length: 8 km, supplying 11 MV/LV sub-stations whose rated power is 250 kVA;
3. two lines of C type, mixed cable-overhead lines, total length: 21 km, supplying 22 MV/LV sub-stations whose rated power is 250 kVA.

In this work we consider only those features that are monitored on one of the lines of type C. The faults considered occur on the lines of type C and type A only. The simulation outputs ten electrical features, listed in Section 9.2, at seven control points. Therefore the application involves seventy features in total. The number of events to be distinguished is $e_t = 31$. The same faults occurring in two different sections are considered as two different faults. We will specifically consider six different working conditions:

1. Normal operating condition.
2. Insulation breakdown (single-phase and double single-phase).
3. Mechanical breakdown (direct and inverse fault).
4. Three-phase fault.
5. Phase-to-phase fault.
6. Phase-to-phase-to-ground fault

A fault event on a line of type A, different from that where the simulated measurements are currently performed is not included here. It can indeed be identified by measurements at the control points at the start of each of the lines. In this case, analytical studies on the network (Di Silvestre 1998) have proved that measurements of the negative component of the reactive power and of the positive component of voltage or current in symmetrical and non-symmetrical systems can tell us whether the line under observation or an external line is in one of the following states.

1. Normal operation.
2. Three-phase fault.

3. Insulation breakdown, mechanical breakdown, two-phase, and two-phase-to-ground faults.

Once the faulty line has been identified, the feature selection for precise fault identification and location detection can be performed. This is the object of the present study. The events we are considering here can take place on one of the main feeders of type C, in which the metering systems are placed at each MV/LV sub-station. Therefore, the size of the search space is 2^f. Since f linearly grows with the number control points c, the search space grows exponentially with both the number of control points and the number of features.

Finally, due to the flexibility of the tool used for feature selection, the present study can easily be extended to include different possible implementations for various different objective function formulations. Of course, the ranges of variation of the features are affected by errors due to the model used. Also, when any diagnosis is actually to be performed, the possible errors due to the measurement systems themselves have to be considered in the diagnostic model formulation. The following section describes the formulation of the objective function.

10.1 The objective function formulation

The objective is to minimize the number of features required for the identification of all events. To explain the criterion used for the objective function implementation a simple case is reported in Fig. 3.14.

For example, consider a system with four features and three possible events, so $f = 4$ and $e_t = 3$; in this case, we choose $c = 1$. Figure 3.14(a) shows the ranges of variation for the given set of features, whereas Fig. 3.14(b) is the table resulting from the comparison of the ranges in Fig. 3.14(a). The

Features → / Events ↓	F_1	F_2	F_3	F_4
E_1	11078	3.9	12.8	33
	11115	54.8	538.4	65.5
E_2	11078	3.7	12.7	66
	11157	116	1120.7	66.5
E_3	11074	0.6	83.4	66.2
	11075	3	330.1	66.7

a)

	F_1	F_2	F_3	F_4
$E_1 E_2$	0	0	0	1
$E_1 E_3$	1	1	0	1
$E_2 E_3$	1	1	0	0

b)

Fig. 3.14 (a) Table containing the ranges of variation of features F_1–F_4 relating to the difference events E_1–E_3; (b) Table containing binary values indicating the pairwise status of the ranges (overlapping = 0; not overlapping = 1).

number of rows is 3 and it comes from equation 5 with $e_t = 3$. As can be seen, none of the four features is, by itself, able to distinguish all events. Only the tuples (F_1, F_4) and (F_2, F_4) are able to do that. Moreover, all the sets containing these tuples are able to distinguish all the events, but they are not minimal sets. Since we are interested in finding the solution that distinguishes all events from one another with a minimal number of features, we can define the objective function to be,

$$\max\left(e_i + 1 - \frac{f_u^2}{f}\right) \tag{6}$$

where e_i is number of events that can be distinguished by the features contained in the solution set and f_u is the number of features in the solution set. The following section presents the experimental results.

10.2 Results

The diagnostic system design problem is an important class of problem in electrical engineering. In addition to finding a good solution comprised of the optimal feature set, the physical interpretation of the solution is also important. Since the GEMGA detects the linkage sets that correspond to the physical dependencies among the features, the linkage information is also valuable.

In such an application GEMGA provides a good insight into the physical problem. The GEMGA searches for linkage sets among variables, which can indicate the contribution of sets of electrical features to the diagnosis problem.

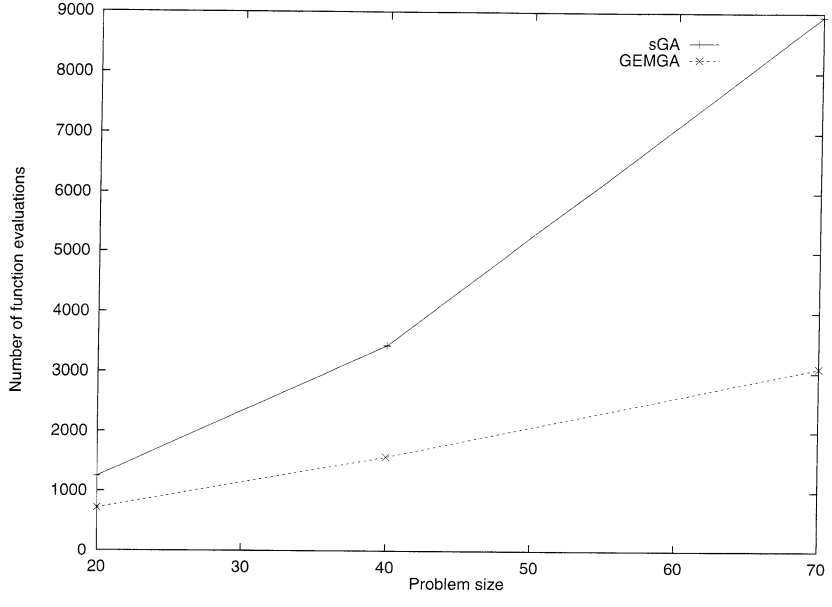

Fig. 3.15 Performance of the GEMGA. Note that the number of function evaluations grows linearly with the problem size.

Feature combinations are represented as *chromosomes*, with each *gene* taking a boolean value. A zero and a one represent absence and presence of the corresponding feature respectively. The *chromosomes* are evaluated according to equation 6. Both the simple GA and the GEMGA are applied separately to solve the problem. Five independent runs are considered and the average number of function evaluations per success are computed for both the SGA and the GEMGA. Figure 3.15 shows the growth of the number of function evaluations with respect to increasing problem size.

As we note, the GEMGA outperforms the SGA by a large factor. Moreover, the increase in the number of function evaluations for the GEMGA is linear, as predicted from the theoretical analysis. The reference minimal solution set has been determined by execution of several runs on each problem size. The optimal solution for the seventy, forty, and twenty features problem contains eight, five, and three features respectively.

These results have a physical explanation. Since the solution set should be able to detect the fault type besides its location, the systems considered have indeed seven, four, and two control points for the seventy, forty, and twenty features. One feature in all the solution sets is the positive current at the start of the line. This gives evidence of symmetrical fault occurrence and location all over the line. The remaining features give evidence of non-symmetrical fault identification and location. These are mostly negative components and zero components of active and reactive power and are good indicators of non-symmetrical faults in symmetrical systems. The linkage sets detected by the GEMGA offers some insights. For example, a typical GEMGA run would find a linkage set involving the inverse component of voltage and of current at the same control point. Moreover, the schema detected by the GEMGA over this linkage partition contains a one for the voltage and a zero for the current. This is naturally expected, since for this application, in most cases these features are mutually exclusively physically.

Further developments of the present study will take into account the non-symmetry of the system. They will also consider the uncertainty arising due to measurement errors and the approximations of the used model.

11. Conclusions

Data mining is an important application area and genetic algorithms are likely to continue their popular role as an effective data mining tool. However, this chapter points out that scalability of genetic algorithms is likely to play an important role in the success of GAs in the twenty-first century. This chapter focused on this scalability issue and presented at least two important fundamental concepts. Like any black-box search/optimization algorithm, GAs are fundamentally based on an inductive search process. Therefore, searching for appropriate relations and classes defined over the search space members is important for transcending the limits of enumerative search. This chapter has presented one possible approach

towards efficient detection of relations and classes by identifying the approximate symmetry-preserving and symmetry-breaking dimensions. It has been shown (Kargupta 1996*a,b*; Bandyopadhyay *et al.* 1998; Kargupta and Bandyopadhyay 1998) that the GEMGA based approach is promising and it has produced linear-time performance for a large class of problems. In this chapter we extended the linear-time performance to a feature selection problem using electrical power distribution network-fault detection data. We hope that this work takes the decade-long effort on scalable GAs one step closer to the holy grail.

Acknowledgements

This work was supported by the School of Electrical Engineering and Computer Science, Washington State University. The second author would like to thank Dr. Maria Luisa Di Silvestre, Professor Angelo Campoccia, Professor Luigi Dusonchet and Professor Antonino Augugliaro for their collaboration and for furnishing the electrical data.

References

Ackley, D.H. (1987). *A connectionist machine for genetic hill climbing*. Kluwer Academic, Boston.

Bagley, J.D. (1967). The behaviour of adaptive systems which employ genetic and correlation algorithms. *Dissertation Abstracts International*, **28**, 5106B. (University Microfilms No. 68-7556).

Bandyopadhyay, S., Kargupta, H., and Wang, G. (1998). Revisiting the GEMGA: Scalable evolutionary optimization through linkage learning. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (pp. 603–8). IEEE Press.

Bhargava, H.K., and Jacobson, D.L. (1997). Exploratory data analysis with genetic algorithms: is there a gulf war syndrome? In *Proceedings of the Second INFORMS Conference on Information Systems and Technology* (pp. 19–28).

Chang, C., Chen, J. Rinivasan, D., Wen, F., and Liew, A. (1997). Fuzzy logic approach in power system fault section identification. *IEE Procedings Generation Transmission, Distribution*, 406–14.

Cui, J., Fogarty, T.C., and Gammack, J. (1993). Searching database using parallel genetic algorithms on a transputer computing surface. *Future generation Computer System*, **9**, 33–40.

De Jong, K.A. (1975). An analysis of the behavior of a class of genetic adaptive systems. *Dissertation Abstracts International*, **36**, 5140B. (University Microfilms No. 76-9381).

Deb, K. (1991). *Binary and floating-point function optimization using messy genetic algorithms* (IlliGAL Report no. 91004 and doctoral dissertation, University of

Alabama, Tuscaloosa). Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

DeJong, K.A., Spears, W.M., and Gordon, D. (1993). Using genetic algorithms for concept learning. *Machine Learning*, **13**, 161–88.

Di Silvestre, M. (1998). *Modelli di analisi diagnostica delle reti elettriche di distribuzione finalizzati al miglioramento della qualita' del servizio*. PhD. Thesis, University of Palermo, Italy.

Frantz, D.R. (1972). Non-linearities in genetic adaptive search. *Dissertation Abstracts International*, **33**, 5240B-41B. (University Microfilms No. 73-11,116).

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, New York.

Goldberg, D.E. and Bridges, C.L. (1990). An analysis of a reordering operator on a GA-hard problem. *Biological Cybernetics*, **62**, 397–405. (Also TCGA Report No. 88005).

Goldberg, D.E., Korb, B., and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, **3**, 493–530. (Also TCGA Report 89003).

Goldberg, D.E., and Lingle, R. (1985). Alleles, loci, and the travelling salesman problem. In Grefenstette, J.J. (Ed.), *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (pp. 154–159).

Greene, D.P., and Smith, S.F. (1993). Competition-based induction of decision models from examples. *Machine Learning*, **13**, 229–57.

Greene, D.P., and Smith, S.F. (1994). Using coverage as a model building constraint in learning classifier systems. *Evolutionary Computation*, **2**, 67–91.

Harik, G. (1997). *Learning linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. Doctoral dissertation, Department of Computer Science, University of Michigan, Ann Arbor.

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor.

Hsu, Y., Lu, F., Chien, Y., Liu, J., Lin, J., Yu, H., and Kuo, R. (1991). An expert system for locating distribution systems faults. *IEEE Trans. On Power Delivery*, **6**, 366–71.

Kargupta, H. (1995, October). *SEARCH, Polynomial Complexity, and The Fast Messy Genetic Algorithm*. Doctoral dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. Also available as IlliGAL Report 95008.

Kargupta, H. (1996a, July). *Computational processes of evolution: The SEARCH perspective*. Presented in SIAM Annual Meeting, 1996 as the winner of the 1996 SIAM Annual Best Student Paper Prize.

Kargupta, H. (1996b). The gene expression messy genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (pp. 814–19). IEEE Press.

Kargupta, H. (1998). *Gene expression and large scale evolutionary optimization*. Kluwer Academic Publishers.

Kargupta, H., and Bondyopadhyay, S. *A perspective on the foundation and evolution of the linkage learning genetic algorithms*. In communication.

Kargupta, H., and Bondyopadhyay, S. (1998). *Further experimentations on the scalability of the gemga*. In press.

Kargupta, H., and Goldberg, D.E. (1996). SEARCH blackbox optimization, and sample complexity. In Belew, R., and Vose, M. (Eds.), *Foundations of Genetic Algorithms* (pp. 291–324). Morgan Kaufmann, San Mateo, CA.

Levenick, J.R. (1991). Inserting introns improves genetic algorithm success rate: Taking a cue from biology. In Belew, R.K., and Booker, L.B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 123–127). Morgan Kaufmann, San Mateo, CA.

Mitchell, M. (1996). *An introduction to genetic algorithms* (1st ed.). MIT Press, USA.

Mitchell, T.M. (1980). *The need for biases in learning generalizations* (Rutgers Computer Science Tech. Rept. CBM-TR-117). Rutgers University.

Neri, F., and Giordana, A. (1995). A parallel genetic algorithm for concept learning. In *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 436–43). Morgan Kaufmann, San Mateo, CA.

Paredis, J. (1995). The symbolic evolution of solutions and their representations. In Eshelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 359–65). Morgan Kaufmann, San Mateo, CA.

Punch, W.F., Goodman, E.D., Pei, M., Chia-Shun, L., Hovland, P. and Enbody, R. (1997). Further research on feature selection and classification using genetic algorithms. In *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 557–64). Morgan Kaufmann, San Mateo, CA.

Radcliffe, N. (1995). *Ga-miner: Parallel data mining with hierarchical genetic algorithms final report* (Technical Report EPCC-AIKMS-GA-MINER-REPORT 1.0). Quadstone Ltd.

Rosenberg, R.S. (1967). Simulation of genetic populations with biochemical properties. *Dissertation Abstracts International*, **28**, 2732B. (University Microfilms No. 67-17,836).

Schaffer, J.D., and Morishima, A. (1987). An adaptive crossover distribution mechanism for genetic algorithms. In Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 36–40).

Smith, J., and Fogarty, T. (1996). Recombination strategy adaptation via evolution of gene linkage. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (pp. 826–31). IEEE Press.

Smith, S.F. (1980). A learning system based on genetic adaptive algorithms. *Dissertation Abstracts International*, **41**, 4582B. (University Microfilms No. 81–12638).

Smith, S.F. (1983). Flexible learning of problem solving heuristics through adaptive search. *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, 422–5.

Smith, S.F. (1984). Adaptive learning systems. In Forsyth, R. (Ed.), *Expert Systems: Principles and Case Studies* (pp. 169–89). Chapman and Hall, New York.

Teo, C. (1995). Machine learning and knowledge building for fault diagnosis in distribution network. *Electric Power and Energy Systems*, **17**, 119–22.

Thierens, D., and Goldberg, D. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). Morgan Kaufmann, San Mateo, CA.

Togami, N., Abe, T., and Kitahashi (1995). On the application of a machine learning technique to fault diagnosis of power distribution lines. *IEEE Trans. on Power Delivery*, **10**, 1927–36.

Watanabe, S. (1969). *Knowing and guessing–A formal and quantitative study*. John Wiley & Sons, Inc, New York.

Wen, F., and Chang, C. (1998). Probabilistic approach for fault-section estimation in power systems based on a refined genetic algorithm. *IEE Procedings Generation Transmission, Distribution*, **144**, 160–8.

Zhu, J., Luckeman, D., and Girgis, A. (1997). Automated fault location and diagnosis on electric power distribution feeders. *IEEE Trans. on Power Delivery*, **12**, 801–9.

4 Theory and application of fuzzy methodology

Paul P. Wang and Fuji Lai

1. Introduction

The challenge of extracting information from a knowledge base is a very important practical issue in AI research. There have been numerous proposals in the past as to how to approach this problem. At the same time, other research has demonstrated in fuzzy control problems that fuzzy logic is an effective tool for solving practical problems. In this chapter we demonstrate that fuzzy set theory is a valuable methodology for the design of knowledge bases, which is a task closely linked to information extraction.

The issues of inference and the organization of a knowledge base are very much intertwined. In consequence, the design of a knowledge base must consider the mechanism of inference at the earliest stages. Traditional AI methods for making inferences, such as rule-based knowledge bases and semantic networks, have proven to be less effective than fuzzy logic methods using approximate reasoning.

Classification is fundamental to the organization and to the efficient and effective use of knowledge. The task of classification exists in all aspects of life and in all academic disciplines (Nilsson 1965). For example, in chemistry the Periodic Table is used as an ordered classification of elements. In zoology classifications exist of different families of animals. For this specific classification problem there is no unique way to proceed. Taxonomy is like language in that it comprises syntactic and semantic aspects. In a given taxonomy task, it is desirable not only to completely separate sub-classes (syntactic), but also to attach meaning (semantic) to each classification. In this chapter we outline how classification tasks can be accomplished according to so-called 'features'. We will illustrate this using an example from industry, in which automated classification of small manufactured items can have far-reaching consequences in efficiency, to demonstrate how classification can be achieved with the use of fuzzy set theory (Kaufman 1975).

In pursuing the task of classification, one must first focus on the issue of pattern recognition. Human cognitive processes have been shown to use feature extraction and analysis in recognizing and classifying patterns or objects. Other theories in cognitive psychology have suggested instead a template-matching method of cognitive recognition, but these have been unable to account for the remarkable flexibility of human powers of recognition, whereby recognition is still possible in the face of adverse conditions such as translation, enlargement, rotation, or segmentation of patterns (Schalkoff 1992; Duda and Hart 1973). Pattern recognition is the first step to making a connection between the real world and the digital computer,

and therefore is vital in tackling real world engineering problems with a computer.

One must develop numeric descriptions and data structures for patterns, because this is the nature of a digital computer. One must be able to pick a pattern or sample and generate data to be used for recognition. Intensive research is needed to translate data from realistic data structures to features. Our research has demonstrated how sample classification and recognition can be achieved through the creation of a knowledge base of features, and further inferences made using this knowledge base. The key tools are fuzzy set theory, membership functions, fuzzy relations, the properties of similarity relation matrices, and fuzzy inference. Even 'approximate reasoning' can be employed advantageously in resolving ambiguous cases.

2. Cognitive science

Cognitive psychology refers to all processes by which 'sensory input is transformed, reduced, elaborated, stored, recovered and used' (Best 1986). Our cognitive processes respond to sensory input and transform physical energy into codes of natural or neural energy. The nervous system receives a physical stimulus and encodes it in such a way as to preserve many of its characteristics. Not only is a neural code created, but so is a cognitive code. This kind of code can be defined as 'the transformations of physical energy that are potentially capable of entering our awareness, or those transformations that form the basis of such an event' (Best 1986). Once created, a cognitive code is reduced or elaborated. An example of the creation of a cognitive code is the process of reading, which involves the assimilation of the meanings of each word to assemble a pattern of meaning. Later, one might be hard-pressed to remember the exact words of the passage read, but one would be able to recall the general meaning. The processes involved in the storing, transforming, recovering, and reconstructing cognitive codes form the basis of our mental lives and what is often referred to as memory.

Of importance in visual perception and object recognition is the process of pattern recognition: the template-matching theory and the feature analysis theory. The first of these is based on the assumption that a faithful retinal image of the object is transmitted to the brain, where it is compared with a stored knowledge base of patterns called templates (Pao 1989). For example, in letter recognition, the perceptual system tries to compare the input letter to the stored templates to achieve a match between the retinal pattern created by the letter **A**, for example, and the retinal pattern template. A practical example is the computerized check-sorting machine used by banks to read account numbers printed on checks. For this to succeed, nearly uniform size and position of the printed characters is required. Template matching theory has two disadvantages. One is that the system is inefficient, requiring much time to sort through many templates and compare them with an image. The second is that pattern recognition in humans is very flexible, as we can recognize the same characters even if they are displaced, rotated, enlarged, or blurred, and this suggests that template-matching is perhaps not the method the brain uses.

Feature analysis has as its basic assumption that all complex stimuli are composed of distinctive and separable parts known as features. Pattern recognition occurs by noting the presence or absence of features and comparing the count with a tabulation of features linked with different labels. This relies upon 'decomposability of stimuli' (Best 1986). Such a tabulation has been shown to be true for letters. The features of the alphabet consist of horizontal lines, vertical lines, lines at approximately 45 degrees and curves. According to these features, a letter **A** may be seen to consist of two lines at 45 degree angles and a horizontal line. Some have suggested that this theory is not such a departure from the template-matching theory, saying that features are only mini-templates. Nevertheless, feature analysis theory is more versatile since the features are simpler, so difficulties encountered by template models may be reduced or eliminated. Also, the feature-combination scheme allows recognition of the relationships most critical to the pattern, as in the case of the letter **A**, where the crucial characteristic is the two 45 degree lines meeting at a point and intersected by a cross bar. Thus the letter A can be recognized in all its possible presentations. The feature theory also is more efficient due to the smaller number of templates required, as templates are not needed for every possible pattern but only each feature. Experiments have established that there is much validity in the feature analysis theory. For example, in letter recognition tasks, it has been noted that letters with many features in common, such as with **C** and **G**, are more prone to be confused.

Feature analysis models gained popularity during the 1960s when physiologists posited 'feature detectors' in the visual cortex of cats and monkeys (Bruce 1990). In Selfridge's Pandemonium system (1959), originally devised for Morse Code recognition but later popularized in alphanumeric recognition (Bruce 1990), pattern recognition takes place in a series of stages carried out by highly specialized cognitive processes, known as 'demons'. First, image demons convert the physical stimulus into a cognitive form recognizable by other demons. The feature demons analyze this information, each scanning for its own particular feature. Next, the cognitive demon checks for its own particular set of features. Every appropriate feature seen increases the noise-making of the demon by one notch. In the final stage, the decision demon listens to the resulting pandemonium, decides who is screaming the loudest and determines that that is the incoming stimulus. A Pandemonium system can also learn to give different weights to different features according to the discriminating power of these features. Although this model is interesting, it was proven unsatisfactory as an absolute model for human recognition because, although it classified patterns, it failed to retain and describe the differences between patterns of the same class.

The most valuable lesson to be learnt from the feature analysis theory is the general and flexible representational format for human pattern and object recognition provided by structural descriptions (Bruce 1990). A structural description is not a theory but provides the right kind of representation with which to construct a theory. It consists of a set of symbolic propositions which define the structural arrangements of the parts. For example, the letter **T** may be defined as a horizontal line supported by a vertical line with the support occurring about halfway along the horizontal line.

The extraction and recognition of features has been established as the most vital component of pattern and object recognition. In the identification of the 25 samples, discussed later in this chapter, we have simulated the human cognitive processes which go into making such a decision. In our experiment eight features have been extracted to construct a knowledge base which can be used to identify objects. As in the whimsical pandemonium model, there are 'demons' or features which have higher distinction powers.

Many artificial systems have been developed to attempt to classify patterns. One, the WISARD (Wilkie, Aleksander and Stonham's Recognition Device) System demonstrated how instead of using features, a neural network which stored responses to a large number of instances of different pattern exemplars was able to classify new patterns correctly (Anderson 1985). This system relied on sampling of select n-tuples of pixels and analysis of statistical similarity. Although this system might play a role in industrial sorting applications, it is limited because results vary according to the lighting. WISARD demonstrates that sorting can be done by brute force and the use of extensive amounts of memory, but awareness of the abstract characteristics of objects would help solve the task in a more intelligent and efficient way.

3. Fuzzy set theory and similarity relation matrices

Fuzzy set theory was introduced by Lofti Zadeh in 1965 as a way of representing the vagueness in everyday life. It is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth. Rather than being regarded as a single theory, it has been said that one should approach the process of 'fuzzification' as a methodology to generalize any specific theory from a crisp (discrete) to a continuous (fuzzy) form.

The essence of fuzziness closely resembles the nature of human cognitive processes (Zadeh 1975). For example, in everyday language one would advise that a driver 'apply the brakes soon', rather than 'begin braking 74 feet from the crosswalk'. The latter instruction is too precise to be implemented. In all aspects of daily life, one learns to assimilate and live by fuzzy data, vague rules, and imprecise information (Bezdek and Pal 1992). Thus, it is reasonable to think that computational models of real systems should be able to recognize and interpret fuzzy uncertainties (Wang and Chang 1981; Wang 1981, 1983, 1993).

Just as there is a strong relationship between Boolean logic and the concept of subject, there is a similar relationship between fuzzy logic and fuzzy subset theory. In classical set theory, a subset U of a set S can be defined as a mapping from the elements of S to the elements of the set {0,1}:

$$U : S \rightarrow \{0,1\}$$

This mapping can be represented as a set of ordered pairs, with exactly one ordered pair present for each element of S, and the first element of the ordered pair being an element of S, the second element being either 0 or 1. The value zero represents total non-membership and the value one represents membership. The truth or falsity of the statement

$$x \text{ is in } U$$

is determined by finding the ordered pair whose first element is x. The statement is true if the second element of the ordered pair is 1, and the statement is false if it is 0.

In a parallel manner, a fuzzy subset F of a set can be defined as a set of ordered pairs, each with the first element from S, and the second element from the interval [0,1], with exactly one ordered pair present for each element of S. This defines a mapping between elements of the set S and values in the interval [0,1]. Zero represents complete non-membership while the value one represents complete membership, and values in between are used to represent intermediate degrees of membership. The set S is referred to as the 'universe of discourse' for the fuzzy subset F. The mapping may be described as 'the membership function' of F. The degree to which the statement

$$x \text{ is in } F$$

is true is found by finding the ordered pair whose first element is x. The degree of truth of the statement is the second element of the ordered pair. There are many forms of membership functions. For example, the concept of 'tallness' is affected by linguistic perceptions as well as cognitive perceptions. If we define a fuzzy subset TALL, the question will become 'to what degree is person x tall'? Each person would be assigned a membership grade defining his 'tallness'. One way in which this could be done is a membership function according to height.

$$\text{tall}(x) = \begin{cases} 0: & \text{if height } (x) < 5 \\ (\text{height } (x) - 5 \text{ ft})/2 \text{ ft} & \text{if } 5 \text{ ft} < \text{height } (x) < 7 \text{ ft} \\ 1: & \text{if height } (x) > 7 \text{ ft} \end{cases}$$

The graph would then be of a triangular form. Membership functions can also exist in other shapes, such as bell curves or staircase graphs. They may be derived from sources such as subjective evaluation or data and probabilities.

To manipulate fuzzy sets, we need operations to combine them. 'Classical' operations were laid out by Zadeh (1975) and are characterized in terms of F(X) = All Fuzzy Subsets of X and m ϵ F(X) $\Leftrightarrow$ m : X $\rightarrow$ [0,1]. The fuzzy sets m_A, m_B ϵ F(X) and the operations are:

(=) Equality $A = B \Leftrightarrow m_A(x) = m_B(x)$
($\subset$) Containment $A \subset B \Leftrightarrow m_A(x) \leq m_B(x)$
($\sim$) Complement $m_A = 1 - m_A(x)$
($\cap$) Intersection $m_{A \cap B}(x) = min\{m_A(x), m_B(x)\}$
($\cup$) Union $= m_{A \cup B}(x) = max\{m_A(x), m_B(x)\}$

We could develop the concept of tallness by relating it to a pair of variables such as the person's height and the person's age, giving a two-dimensional membership function or 'fuzzy relation'. In pattern or object recognition, the aim is to search for structure in data. One way to approach this is to use the concept of fuzzy sets to transform input data into a set of membership values which indicate the degree of membership in certain classes and the degree of similarity between different classes. We now discuss how the notion of fuzzy relations and fuzzy similarity relations can give an indication of the similarity of the two samples.

An example of a fuzzy relation is shown below, where we let P be a product set of n sets and M its membership function, and a fuzzy n-ary relation is a fuzzy subset of P taking its values in M (Wang 1976).

$$E_1 = \{x_1, x_2, x_3\},\ E_2 = \{y_1, y_2, y_3, y_4, y_5\},\ M = [0,1]$$

	y_1	y_2	y_3	y_4	y_5
x_1	0	0	0,1	0,3	1
x_2	0	0,8	0	0	1
x_3	0,4	0,4	0,5	0	0,2

In the case where $E_1 = E_2 = E$ and $M = [0,1]$ the result is a fuzzy binary relation. The membership grade associating each pair gives the strength of the relation between members. Similarity relation matrices or relations of similitude are a subset of fuzzy binary relations which have the property of 'equivalence'. Within equivalence are three requirements that must be satisfied:

(a) Symmetry. A symmetric fuzzy binary relation is defined by

$$\forall(x,y) \in \mathrm{E} \times \mathrm{E}: (\mu_R(x, y) = \mu) \rightarrow (\mu_R(y, x) = \mu).$$

(b) Reflexivity. This property is defined by

$$\forall(x, y) \in E \times E: \mu_R(x, x) = 1$$

(c) Transitivity. If we let $x, y, z \in \mathrm{E}$; then

$$\forall(x, y), (y, z), (x, z), \in E \times E:$$
$$\mu_R(x, z) \geq \mathrm{MAX}\,[\mathrm{MIN}\,(\mu_R(x, y), \mu_R(y, z))].$$

An example of a fuzzy similarity relation is shown below and it can be seen to satisfy all three requirements. The properties of symmetry and reflexivity can be rationalized by recognizing that the similarity of one sample to itself should be one and the similarity of two samples should be the same regardless of the order in which they are taken.

R	A	B	C	D	E
A	1	0,1	0	0,1	0,9
B	0,1	1	0,2	0,3	0,4
C	0	0,2	1	0,5	0,2
D	0,1	0,3	0,5	1	0,1
E	0,9	0,4	0,2	0,1	1

Thus the entries on the diagonal are unity and those on one side of the diagonal are reflected onto the other side. These similarity relation matrices can be used to form inferences and putative identifications. This can be done through several different methods such as max-min composition, max-star composition, or max-product composition (Kaufman 1975). The max-min composition is defined such that if $R_1 \subset X \times Y$ and $R_2 \subset Y \times Z$, then the min-max composition R_1 and R_2 denoted R_2 o R_1 is:

$$\mu_{\mathrm{R2oR1}}(X, Z) = V_y\,[\mu_{R1}(\mathrm{x}, \mathrm{y}) \wedge \mu_{R2}(x, y)] = \mathrm{MAX}_y\,[\mathrm{MIN}\,(\mu_{R1}(x, y)\ \mu_{\mathrm{R2}}(y, z))]$$

where $x \in X$, $y \in Y$, and $z \in Z$. An example is shown below:

					R	y_1	y_2	y_3	y_4
x_1	x_2	x_3	x_4		x_1	1	0,8	0,3	0,4
0,3	0,4	0,5	0,1	∘	x_2	0,8	1	0,2	0,5
					x_3	0,3	0,2	1	0,6
					x_4	0,4	0,5	0.6	1

The operation is carried out by taking the 1×4 matrix in turn with each column of the 4×4 matrix in the manner:

$$\text{MAX}\,[(0{,}3 \wedge 1), (0{,}4 \wedge 0{,}8), (0{,}5 \wedge 0{,}3), (0{,}1 \wedge 0{,}4)] =$$

$$0{,}3 \vee 0{,}4 \vee 0{,}3 \vee 0{,}1 = 0{,}4$$

(Note: $\vee$ = minimum; $\wedge$ = maximum)

The result would be: [0,4 0,4 0,5 0,5]

In this manner inferences can be made. There exist at least 12 different ways of making inferences and max-min composition is not the only method possible (Togai and Wang 1982, 1984, 1985).

4. Knowledge base

The creation of a knowledge base must precede all other steps in any machine intelligence task. In describing different samples or patterns, features must be extracted first; identification can proceed later with the use of those features in a knowledge base and a decision-making policy or discrimination functional block (Watanabe 1985; Pao 1989).

Let us consider Boolean logic in which the binary variable '1' symbolizes a 'yes' and a '0' symbolizes a 'no'. To distinguish two samples at least one bit of information is needed, and for four different samples at least two bits of information are needed. Thus a minimum of n features are needed for identification if the number of samples is $2n$. The detailed selection of features is important to optimize efficiency and discrimination powers. If features are chosen correctly, both the number required for identification and computation time are kept at a minimum (Bruce 1990).

One advantage of fuzzy set and fuzzy logic theory as opposed to Boolean logic or crisp set theory is that variables need not be assigned to only zero or one, but instead may take any real value in this range (Zadeh *et al.* 1975). In an ideal situation, one feature would be sufficient to distinguish between any number of samples, as the membership grade would be able to take on any value between 0 and 1. However, this situation exists only if there is such a perfect feature that each sample has a different grade valuation in the membership function relation representing that feature.

In reality one feature is usually insufficient for identification as there is the problem of spacing between membership grades of samples. It is advantageous to have larger spacing because this raises the tolerance to noise. In any

realistic situation, noise is a force to be reckoned with, and in our particular problem noise may arise in the translation from real three dimensional sample to two dimensional signal (image), and the eventual mapping to single numeric scalar number. The greater the number of features used, the higher will be the accuracy of recognition. Thus it is necessary to experiment and generate as many effective features as possible to yield membership grades with high discrimination power. This is particularly important during the initial stage of design of the recognition system.

The selection of membership function is also not arbitrary. The type of function used was a convex function and basically piecewise linear in nature. In our experiment, 25 industrial samples of washers and nuts were chosen as the patterns to be classified and recognized. To identify all 25 samples, at least five bits of information are required, according to information theory. The extraction of features from the three dimensional samples was done by first relating each sample to a two dimensional image of the head of the sample (Wang 1976; Thint *et al.* 1993; Sollberger *et al.* 1989). Next, one dimensional numbers representing features of the two dimensional images were obtained. These numbers were chosen as ratios which would give an

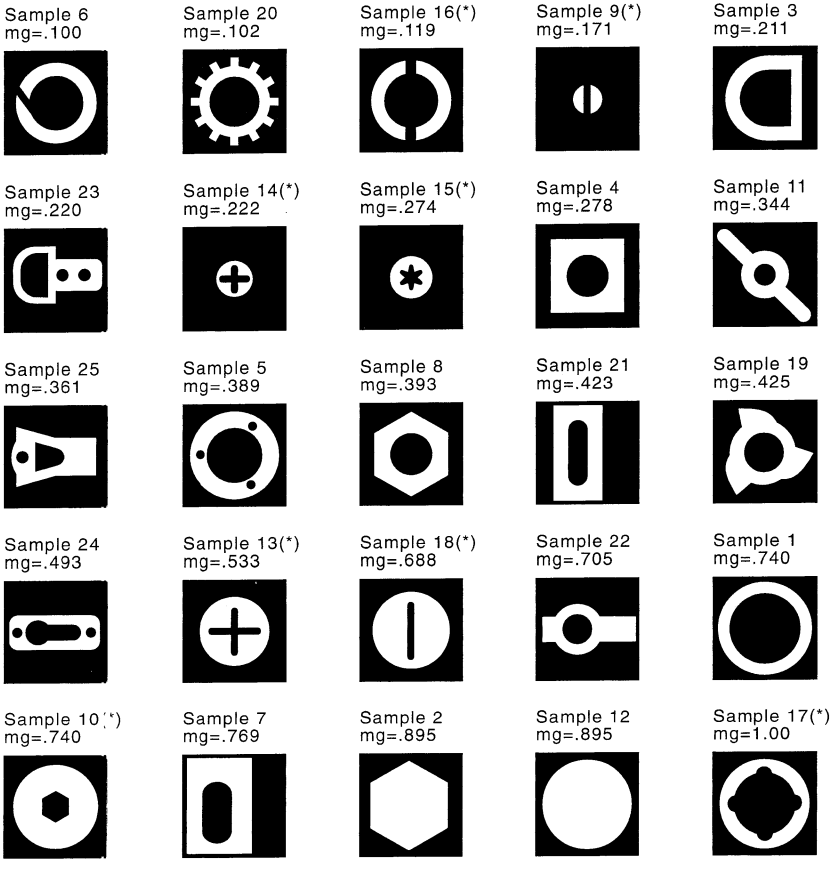

Fig. 4.1 25 patterns for feature 1.

indication of the important characteristics of the sample, such as the general size, the number of sides/edges, and the number of perforations in the sample. Eight such features were selected after much experimentation. The use of eight features should provide better results than just five, but it must be noted that there is a redundancy of information as not all features would present orthogonal basis vectors in a vector space. This redundancy naturally renders our experimental model less than ideal in terms of the spanning of a vector space. On the other hand, this heuristic scheme has the advantage of achieving more accurate recognition rates, taking advantage of majority voting.

For each feature chosen, the samples were renumbered according to their membership grades, from the lowest to highest (typical classification are shown in Figs 4.1 and 4.2).

As mentioned previously, the ordering of the membership grade is important, because one can make 'semantic' interpretations of a specific feature. Next, the membership grade for each sample was compared in turn with each of those of the other samples and the difference was mapped using a convex membership function onto another number in the range [0,1]. These results, representing the degrees of similarity between two chosen samples, were

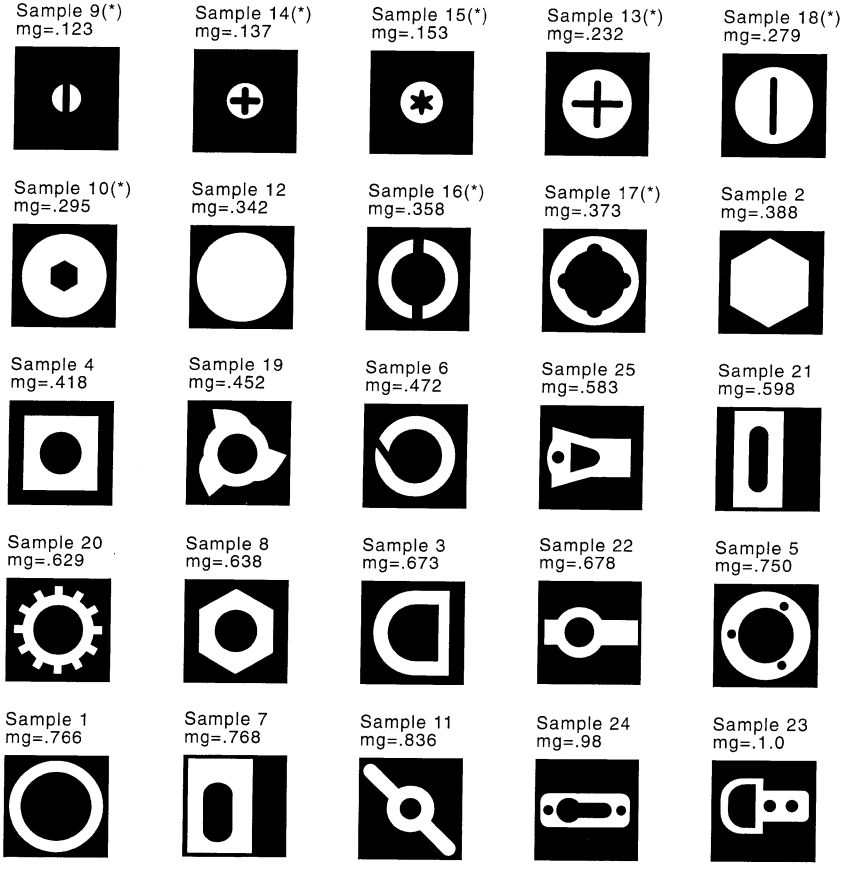

Fig. 4.2 25 patterns for feature 2.

Similarity relation matrix for feature 1

```
1.0.99.99.90.78.75.74.55.53.30.26.19.18.12.12.05.02.00.00.00.00.00.00.00
.991.0.99.91.79.76.75.55.54.31.26.19.18.13.12.05.02.00.00.00.00.00.00.00
.99.991.0.95.84.82.81.62.60.36.31.23.22.16.15.06.03.00.00.00.00.00.00.00
.90.91.951 0.97.95.95.81.79.55.49.39.37.28.28.13.07.00.00.00.00.00.00.00
.78.79.84.971.0.99.99.92.91.70.64.53.52.41.40.20.13.01.01.00.00.00.00.00
.75.76.82.95.991.0.99.94.93.74.67.56.55.44.43.23.14.01.01.00.00.00.00.00
.74.75.81.95.99.991.0.95.94.74.68.57.56.45.44.23.14.01.01.00.00.00.00.00
.55.55.62.81.92.94.951.0.99.91.86.77.75.64.63.38.26.03.02.01.01.01.00.00
.53.54.60.80.91.93.94.991.0.92.87.78.77.66.65.40.27.03.03.01.01.01.00.00
.30.31.36.65.70.74.74.91.921.0.99.96.95.88.88.64.49.09.07.04.04.03.00.00
.26.26.31.49.64.67.68.86.87.991.0.98.98.93.92.71.55.12.09.06.06.04.00.00
.19.19.23.39.53.56.57.77.78.96.981.0.99.98.97.81.66.17.14.09.09.06.01.01
.18.18.22.37.52.55.56.75.77.95.98.991.0.98.98.82.68.18.14.09.09.06.01.01
.12.13.16.28.41.44.45.64.66.88.93.98.981.0.99.91.79.25.20.13.13.09.01.01
.12.12.15.28.40.43.44.63.65.88.92.97.98.991.0.91.79.25.20.13.13.09.01.01
.05.05.06.13.20.23.23.38.40.64.71.81.82.91.911.0.97.47.41.30.30.22.04.04
.02.02.03.07.13.14.14.26.27.49.55.66.68.79.79.971.0.62.55.42.42.33.07.07
.00.00.00.00.01.01.01.03.03.09.12.17.18.25.25.47.621.0.99.95.95.88.42.42
.00.00.00.00.01.01.01.02.02.07.09.14.14.20.21.41.55.991.0.98.98.92.49.18
.00.00.00.00.00.00.00.01.01.04.06.09.09.13.14.30.42.95.981.01.0.98.62.62
.00.00.00.00.00.00.00.01.01.04.06.09.09.13.14.30.42.95.981.01.0.98.62.62
.00.00.00.00.00.00.00.01.01.03.04.06.06.09.09.22.33.88.92.98.981.0.73.73
.00.00.00.00.00.00.00.00.00.00.00.01.01.01.01.04.07.42.49.62.62.731.01.0
.00.00.00.00.00.00.00.00.00.00.00.01.01.01.01.04.07.42.49.62.62.731.01.0
.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.01.01.14.18.26.26.34.80.80
```

Fig. 4.3 The tuned results of the similarity membership function.

entered into a similarity relation matrix. These matrices contain much valuable information which, when presented with information representing an unknown sample, can produce a decision on the identity of the sample. Thus these eight similarity relation matrices form the knowledge base (an example is shown in Fig. 4.3) through which inference will be made.

5. Inference using similarity relation matrices

The ability to make inference is very much tied into human intelligence, which enables a person to apply his powers of identification to newly encountered patterns. The process of inference in our research involves extracting a conclusion in terms of the identity of a sample through use of a similarity relation matrix. A similarity relation matrix is a form of database of features for identification which makes use of closeness properties of patterns. It can also be described in terms of mathematical linguistics as a 'dictionary' whereby identification is carried out by finding the sample's closest match in the dictionary. This matrix can be thought of as a mathematical model which is fixed and ready to use for inference. With the input as real data from an unknown sample, the similarity relation matrix acts as a transfer to output the identity of the sample.

One very important by-product of our research on fuzzy inference is the ability of the model to place a new or foreign sample within our taxonomy system. This is the basic principle of inference. The observation emphasizes the very important notion of 'adaptiveness' as a vital characteristic of machine intelligence. Consequently, the size of the knowledge base will grow. However, the irony is that as a result, the feature space will become progressively tighter and thus more features are eventually required. This also explains the importance of the spacing of the range of the membership grades in feature space.

As in the analogy to a dictionary, a similarity relation matrix can only work perfectly (that is give a definite and correct answer) if ideal data are presented. It represents the measuring stick against which comparisons of similarity will be made. A grade of '1', represents complete similarity while '0', represents complete dissimilarity. In order to maximize accuracy of identification, there are two parameters which can be varied by the designer to fit the situation. These are the type of membership function chosen and the parameter k; optimizing these is known to the fuzzy research community as 'tuning'.

It is important to understand the physical nature of similarity relation matrices in order to appreciate the mechanism of the inference process. Each column and row forms a knowledge base for a particular sample.

The generation of the similarity relation matrices itself involves inference through the calibration of the values of k to obtain optimum results. Let us consider one feature. First the initial membership grades of the row vector of the 25 samples were translated into similarity relation matrix entries using an initial value of k through the method detailed above. To ascertain the appropriateness of this value of k the inference process was carried out using the same ideal sample data. With ideal data providing the input, recognition should be close to 100 per cent.

To demonstrate the process of inference in tuning, let us pick the first sample in the ordering for that particular feature. A 1×25 matrix is created which has entries representing grades of similarity between sample 1 and each of the other samples in turn, e.g. entry 3 is a similarity relation between sample 1 and sample 3. This matrix then undergoes max–min composition with each of the columns of the similarity relation matrix for that feature. The result should be a 1×25 matrix with a 1 in the first position indicating

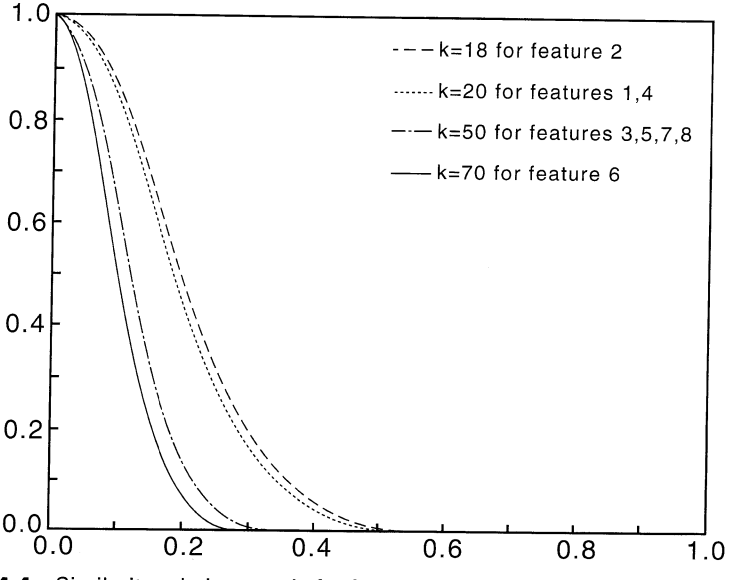

Fig. 4.4 Similarity relation matrix for feature 1.

that the test sample 1 has been correctly identified. This is carried out with each of the other samples for that feature, and the value of k finally chosen for the similarity relation matrix is the one which maximizes recognition accuracy (Fig. 4.4).

Each feature generates different orderings for the samples, and all features do not necessarily have the same optimum k value. During tuning the inference process is essentially one of max–min composition of each of the columns of a similarity matrix with the similarity matrix. In our experiment accuracy was defined through majority voting, according to which a sample was said to have been correctly identified if at least 5 of the 8 features identified it correctly, i.e.. there was a '1' in the correct position for at least five features. If there were '1' entries in more than one position, the sample received a correctness vote of (1 / number of ones) from that feature. The final recognition rate was calculated out of recognition for 25 samples.

The true challenge arises when the similarity relation matrices are faced with data contaminated with noise. The ultimate purpose of the use of the similarity relations pursued in our research is application to real-life automation. We would like to use machine intelligence to carry out sorting tasks for humans. However, noise poses a significant problem since it cannot be avoided in a real industrial environment. Noise in our industrial example may arise from many sources and will have the effect of lowering recognition rate. This issue will be addressed in the next section.

6. Results

Real-life noisy data were simulated to explore the effects of noise on recognition rate. Data were simulated using a random number generator for uniform probability errors of 4 per cent, 8 per cent, 12 per cent, 16 per cent, and 20 per cent as displayed in Fig. 4.5.

(To clarify, an error of 4 per cent means that the integral of the probability of errors from –2 per cent to +2 per cent was 1. The percentages presented here indicate the maximum excursion of the uniform distribution and not statistical parameters such as mean or standard deviation). For each noise rate, 5 trials were carried out. In real applications, noise may arise from many sources, including the translation from three-dimensional object to two-dimensional image (Wang *et al.* 1984; Wang and Fatmi 1986). The random number generator simulated the errors which could possibly infect the values of the raw geometric membership grades obtained.

Twenty-five samples were selected from an array of industrial samples found at the local hardware centre. These samples were used to generate eight similarity relation matrices based on eight features. The properties of two dimensional noise of the industrial parts have not been taken into consideration in a direct manner in this research. However, our simulation of random noise does not hinder the generality of our approach.

Let us begin with the first similarity relation matrix and a noise rate of 4 per cent. The raw geometric membership grade of the first sample was injected with noise of a random value of ± 2 per cent. This value was used as

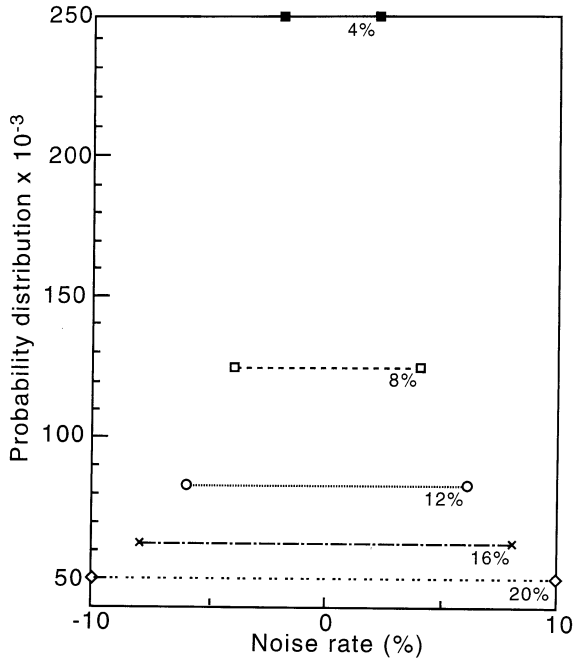

Fig. 4.5 Distribution of noise for each noise rate.

the real-life raw membership value for the first sample, and the inference method described earlier carried out. The noisy membership grade for the first sample was compared in turn with the perfect membership grades for the other samples. The difference was then mapped with the similarity function (using the appropriate tuned k value) defined for that particular feature to give a 1×25 'realistic similarity vector' which represents information about the real-life sample 1. This vector then underwent max–min composition with each of the columns of the similarity relation matrix for that feature to give a 1×25 matrix result. If the '1' was in the correct place, i.e. the first position in this present computation, then recognition was said to have been successful.

This procedure was repeated four times with the same noise, sample and feature, and then a further 5 trials were performed at 8 per cent, 12 per cent, 16 per cent, and 20 per cent noise. The entire process was repeated for this feature for each of the 25 samples. The steps above were repeated for the remaining seven features.

Majority voting was used to calculate recognition accuracy at each noise rate. A sample was said to have been identified correctly if the total number of hits was greater than 20. (The total number of possible hits would be 5 trials $\times$ 8 features = 40). The recognition rate was calculated as the number of such samples out of 25.

As expected, the recognition rate decreased as the percentage of noise increased (Fig. 4.6).

However, it can be seen that recognition is still surprisingly good as the rate was still 100 per cent at 4 per cent error. At 8 per cent and 12 per cent error the rate was relatively high at 88 per cent. As noise increased to 16 per

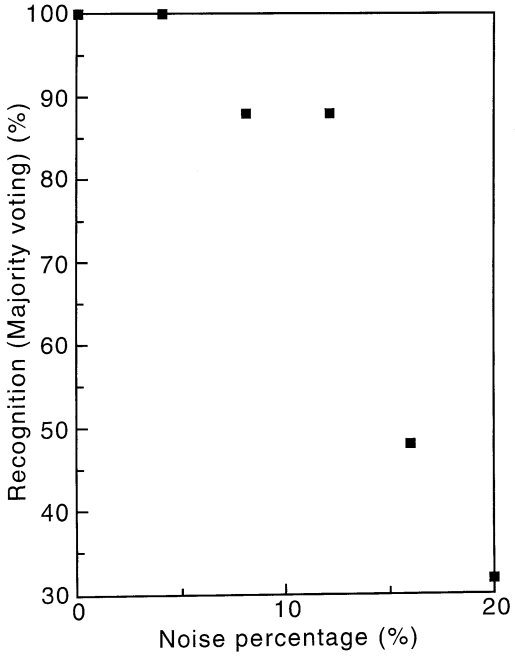

Fig. 4.6 Sample recognition accuracy as a function of noise.

cent and 20 per cent the rates became 48 per cent and 32 per cent respectively. Thus it can be seen that our system was successful in recognizing even in moderate levels of noise.

7. Conclusion

We have shown the richness of the use of fuzzy membership functions as a means of modelling features. This is particularly powerful when coupled with similarity matrices, fuzzy inference, and majority voting. These lead to the successful design of knowledge bases which are vital components in machine intelligence. Furthermore, they can be implemented in hardware to increase the speed or the number of inferences per unit time.

Samples can be identified in even moderate levels of noise and in the face of uncertain or incomplete data. Recognition was still 100 per cent at 4 per cent noise but deteriorated as noise levels increased. This research has established only the lower bounds for the use of the knowledge base and fuzzy inference. There is still much room for exploration and improvement through the process of tuning and design optimization.

In arriving at our conclusion we made several decisions regarding the method to be used. One concerned the injection of noise at the membership grade level. It should be noted that there exists the possibility of future investigation using noise applied at the two-dimensional image stage (Agin 1981; Haralick 1978). Another decision was the use of max–min compositions although several other types of compositions such as the min-product

composition exist (Wang and Wang 1981; Wang and Togai 1985). Further investigation into control of the value of k may give even stronger results. In addition, it may be possible to increase recognition rate by increasing the number of features used. However this has the disadvantage of increasing cost due to the amount of hardware and computation time required. Potentially, we might also be returning to the use of 'brute strength' as opposed to the intelligent selection of features with the highest discrimination power. Usually an optimal combination set of features can be found. The weighting of the multiple features, taking the inner product of the weight vector and the feature vector, may result in high recognition rates; but this would be at the expense of 'semantic interpretation' as it would be difficult to attach a physical meaning to such a membership grade.

Another point of note is that contemporary research on similarity matrices has generated very rich theoretical results (Le 1993, 1994, Tarama *et al.* 197l). These results may lead to further refinement of knowledge base design using fuzzy methodology.

Our research has not touched on all avenues of exploration; prospects for further growth and improvement of this method of recognition are exceptional. The method we have investigated has countless possible applications in science and areas requiring ranking, taxonomy, classification, and recognition (Wang and Fatmi 1986). We are likely to witness the power of feature extraction, the knowledge base, and the similarity relation matrix coming to the forefront of machine intelligence theory in the future.

Acknowledgement: The authors would like to thank Jerry C.Y. Tyan and Hiro Kaneda for their assistance in computer graphics and software. This research was supported in part by National Science Foundation grant No. ECS9216474 and by the Electric Power Research Institute research grant No. EP8030-3.

Appendix: feature generation and selection

The selection of features used for classification and identification was done through extensive experimentation (Wang and Kadonoff 1984; Wang and Ellinwood 1979). Each feature captures a certain aspect of the sample and together they form a dictionary of similarity grades with which to distinguish samples. Not all features have the same discrimination power. We describe below the rationale for the selection and illustrate the method of calculation with an example for each feature.

Feature 1:

This feature attempts to capture the approximate proportions of sample surface area compared with the area of any enclosed hole.

$$\textit{Compact Ratio} = \frac{\textit{Surface Area}}{\textit{Total Edge Length}}$$

e.g. Sample 1
a = 1.5 cm
b = 3.2 cm

$$\text{feature} = \text{compact ratio} = \frac{\pi\left(\frac{b}{2}\right)^2 - \pi\left(\frac{a}{2}\right)^2}{\pi a + \pi b}$$

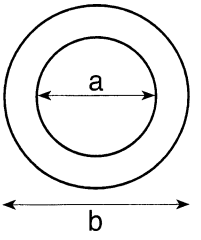

Fig. 4.7

Feature 2:

This feature approximates the edge length of the sample (including outer edges and any inner edges of holes).

e.g. Sample 5
a = 0.4 cm
b = 0.9 cm
c = 2.5 cm
edge length = $\pi(3a + b + c)$

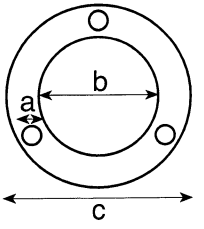

Fig. 4.8

Feature 3:

The surface area gives an idea of the size of the sample.

e.g. Sample 12
a = 2.0 cm
surface area = $\pi(a/2)^2$

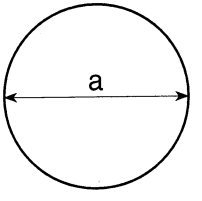

Fig. 4.9

Feature 4:

This was defined as the number of crossings over the surface area. This attempted to extract information about the number of discreet openings in the sample. A crossing was found by centering the sample shape inside the smallest rectangle possible to contain the shape and then drawing lines from the centre of the rectangle to each rectangle edge and rectangle side midpoint. A crossing occurred whenever there was a change from solid to hole or vice versa.

e.g. Sample 4
a = 0.7 cm
b = 1.4 cm

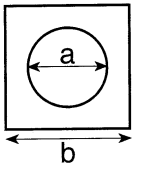

Fig. 4.10

$$\text{feature} = \frac{16}{b^2 - \pi\left(\frac{a}{2}\right)^2}$$

Feature 5:

This gives the ratio of compact ratio over number of crossings.

e.g. Sample

$$\text{feature} = \frac{\textit{compact ratio from feature 1}}{16}$$

Feature 6:

This was defined as the ratio of the compact ratio over the number of sides. A side was defined as any abrupt change in the orientation when following the outer edge of a sample. For instance, a circle would have a side rating of 1 while a hexagon would have a side rating of 6.

e.g. Sample 1

$$\text{feature} = \frac{\textit{compact ratio from feature } 1}{1 \textit{ side}}$$

Feature 7

This feature contains information about the hole-to-surface ratio, the number of holes, and the number of sides of the sample. First the ratio of number of crossings to number of sides was obtained. Then the compact ratio was divided by this number to give the feature.

e.g. Sample 1

$$\text{feature} = \frac{\textit{compact ratio from feature } 1}{\frac{16}{I}}$$

Feature 8:

This was found by taking the ratio of the number of crossings to the number of sides and then dividing by the surface area.

e.g. Sample 12

$$\text{feature} = \frac{\frac{\otimes}{I}}{\textit{surface area from feature } 3}$$

References

Agin, G.J. (1981). Hierarchical representation of three-dimensional objects using verbal models, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **Vol. PAMI-3**, no. 2, pp. 197–204.

Anderson, J.R. (1985). *Cognitive psychology and its implications*. W.H. Freeman & Co., New York.

Best, J.B. (1986). *Cognitive psychology*. West Publishing Co., St. Paul, MN.

Bezdek J.C. and Pal S.K. (1992). *Fuzzy models for pattern recognition*. IEEE Press, New York.

Bruce, V. (1990). *Visual perception*. Lawrence Erlbaum Associates Publishers, Hove, UK.

Duda, R.O. and Hart P.E. (1973). *Pattern classification and scene analysis*. John Wiley and Sons, New York.

Fisher, R.A. (1950). The use of multiple measurement in taxonomic problems, reprinted in *Contributions to mathematical statistics*. John Wiley and Sons, New York.

Haralick, R M. (1978). Structural pattern recognition, homomorphisms, and arrangements, *Pattern Recognition*, **10**, pp. 223–36.

Kaufman, A. (1975). *Introduction to the theory of fuzzy subsets*, Vol. 1, Academic Press, Inc., New York.

Le, K. (1993). A similarity relation and its applications, *Fuzzy theory and technology conference*. Duke University, Durham, North Carolina.

Le, K. (1994). Fuzzy relation compositions and pattern recognition, *Joint conference on information sciences and proceedings abstracts and summaries 1994*. Pinehurst, North Carolina, pp. 409–12.

Nilsson, N.J. (1965). *Learning machines*. McGraw-Hill, New York. (This has been revised as *Mathematical foundations of learning machines*. Morgan Kaufmann, San Mateo, Calif., 1989.)

Pao, Y.H. (1989). *Adaptive pattern recognition and neural networks*. Addison-Wesley, Reading, Mass.

Schalkoff, R. (1992). *Pattern recognition: statistical, structural and neural approaches*. John Wiley and Sons, New York.

Sollberger, D.R., Wang P.P., and Thint M.P. (1989). A flexible inspection system for gauging precision industrial parts. *Robotics and Autonomous Systems*, **5**, 165–71. Elsevier Science Publishers B.V. (North-Holland).

Tarama, S. *et al.* (1971). Pattern classification based on fuzzy relations, in [32], pp. 169–74.

Thint, M.P., Wang P.P., and Dollas A. (1993). Nonparametric graded data processing with back error propagation networks. *Information Sciences*, **67**, Numbers 1 and 2, pp. 167–87.

Togai, M. and Wang P.P. (1982). Sensitivity analysis of dynamic systems via fuzzy set theory. In *Recent developments in fuzzy set and possibility theory*, (Ronald Yager, ed.). Pergamon Press.

Togai, M. and Wang P.P. (1984). Analysis of a fuzzy dynamic system and synthesis of its optimal controller, *Journal of control and cybernetics*, **13**, No. 3. PWN. Polish Scientific Publishers.

Togai, M. and Wang P.P. (1985). *A study of fuzzy relations and their inverse problem* North Holland Publisher.

Wang, P.P. (1976). Machine recognition of quality patterns of an industrial process, conferencia International Y Exposicion Sobre Investigacion Desarollo Y Application de la Ingenieria Electrica Y Electronica, Memoria, Mexico City.

Wang, P.P. (1981). *Fuzzy set theory and applications*. Shanghai Science and Technology Publisher.

Wang, P.P. (ed.), (1983). *Advances in fuzzy sets, possibility theory and applications*. Plenum Publishing Corp., New York.

Wang, P.P. (ed.), (1993). *Advances in fuzzy theory & technology*, Volume 1. Bookwrights Press, Durham, NC.

Wang, P.P. and Chang S.K. (eds), (1981). *Fuzzy sets: theory and applications of policy analysis and information systems*. Plenum Publishing Corp., New York.

Wang, P.P. and Ellinwood E.H., Jr., M.D. (1979). Experiments on behavioural patterns of rats using pattern recognition techniques. In *Proceedings of the international conference on cybernetics and society*. IEEE publication 70CH1424-1 SMC. pp. 914–17 Denver.

Wang, P.P. and Fatmi M. (1986). Fuzzy detection and estimation, In *Communication theory, encyclopaedia of systems and control*. Pergamon Press, Oxford.

Wang, P.P. and Hodgkiss W.S., Jr. (1973). Feature extraction on a finite set of binary patterns. In *Proceedings, international symposium on computers and chinese input/output systems*, pp. 183–94. Academia Sinica, China.

Wang, P.P. and Kadonoff M.B. (1984). Three dimensional object recognition and orientation using a gray-scale tactile sensor, *Proceedings, International Computer Symposium*, Volume I, pp. 617–23.

Wang, P.P., Ma K.K., and Rebman J. (1984). Automatic recognition of low resolution tactile sensing date using rabid transformation, NATO-ASI Series F. (Computer & Systems Sciences) Vol. FII, *Robotics and artificial intelligence*. (ed. M. Brady, L.A. Gerhardt, and H.F. Davidson), pp. 159–70. Springer-Verlag.

Wang, P.P. and Togai M. (1985). Analysis and control of fuzzy dynamic systems, *Man Machine Studies*, **22**, 355–63.

Wang, P.P. and Wang C.Y. (1981). 'Experiment of Character Recognition Using Fuzzy Filters'. In *Fuzzy Sets: theory and applications of policy analysis and information systems*. Plenum, New York.

Watanabe, S. (1985). *Pattern recognition: human and mechanical*. John Wiley and Sons, New York.

Zadeh, L. (1975). *Fuzzy sets and their applications to cognitive and decision processes*. Academic Press, New York.

5 Data representations for evolutionary computation

Ian C. Parmee, Carlos A. Coello Coello, and Andrew H. Watson

1. Introduction

The data processing capabilities of genetic algorithms (GAs) have been recognized (within a wide variety of domains) in recent years, and have received much attention from researchers and practitioners working in many different disciplines (Goldberg 1989). As a stochastic, heuristic technique, the GA does not need specific information about the problem domain to guide search. Its structure is analogous to biological evolution theory using the principle of survival of the fittest (Holland 1975). Therefore, the GA resembles a 'black box' that can be attached to any particular application. In general, the following basic components are required to implement a GA (Michalewicz 1992):

1. A representation for potential solutions to the problem.
2. A way to create an initial population of potential solutions (this is normally done randomly, but deterministic approaches can also be used).
3. An evaluation function that plays the role of the environment, rating solutions in terms of their 'fitness'.
4. Genetic operators that alter the composition of children (normally, crossover and mutation).
5. Values for parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, etc.).

In this chapter, we focus on the first of these components: the representation used by the genetic algorithm.

The traditional representation used to encode a set of solutions is the binary scheme, in which a chromosome is a string of the form $<b_1, b_2, \ldots, b_m>$ (Fig. 5.1), where $b_1, b_2, \ldots, b_m$ are termed *genes* (the values that these genes can assume are called *alleles* and in binary representation are either zeros or ones).

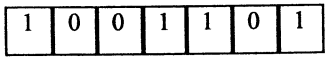

Fig. 5.1 An example of a binary string.

There are several reasons why binary encoding is normally utilized by GAs but most date back to John Holland's pioneering work. In his book, Holland (1975) gave a theoretical justification for the use of binary encoding. He compared two different representations with approximately the same information-carrying capacity, one that had a small number of alleles and long strings (e.g., binary strings of length 80), and the other with a large number of alleles and short strings (e.g., decimal strings of length 24). Notice that 2^{80} (binary) $\approx 10^{24}$ (decimal). Holland (1975) argued that the first encoding allows a higher degree of 'implicit parallelism' than the latter, since it

contains more *schemata* than the second encoding (11^{24} Versus 3^{80}). A *schema* (plural *schemata*) is a template describing a subset of the strings that share certain similarities at some locations across their length (Holland 1975; Goldberg 1989). The presence of more schemata favours diversity, and increases the probability that good 'building blocks' (i.e., the portion of a chromosome that confers higher fitness on the string in which it is present) are formed at each generation, therefore improving the performance of the GA over time according to the schema theorem (Holland 1975; Goldberg 1989). The 'implicit parallelism' of GAs, introduced by Holland, refers to the fact that, while explicitly calculating the fitness of the individuals in a population, the GA implicitly estimates the average fitnesses of a much larger number of chromosomic strings by calculating the observed average fitnesses of the 'building blocks' detected in the population.

Therefore, according to Holland, it is preferable to have many genes with few possible alleles rather than a few genes with many possible alleles. This is not only for theoretical reasons (following Holland's schema theorem), but it also has a biological justification in genetics, where it is more usual to have chromosomes with many genes and few alleles per gene rather than few genes and many alleles per gene. However, the implicit parallelism of GAs does not preclude the use of alphabets of higher cardinality, although a binary alphabet offers the maximum number of schemata per bit of information of any coding (Michalewicz 1992; Goldberg 1989). Nevertheless there has been long debate over these non-binary alphabets, mainly from the practitioner's side.

As we will see in this chapter, the use of a binary representation has several drawbacks when the GA is used to solve real-world problems. For example, if we try to optimize a function with high dimensionality (e.g., 50 variables), and we are interested in good precision (e.g., 5 decimals), then the mapping from real numbers to binary numbers will generate extremely long strings (of perhaps 1000 bits) and the GA will be unable to perform well in most cases unless special operators and procedures are designed for the problem.

In the following pages we will discuss some of the alternative representation schemes that have been proposed to deal with this and with other limitations of the binary representation, providing in each case examples of applications in which such approaches have been found useful.

2. Gray coding

Early work by GA researchers revealed problems in the use of a binary representation and anomalies in the mapping of the search space to the representation space (Hollstien 1971). For example, the integers 5 and 6, which are adjacent in the search space, have binary equivalents of 101 and 110, which differ by 2 bits in the representation space. This phenomenon, known as a *Hamming cliff* (Caruana and Schaffer 1988), has led to alternative representations in which the adjacency property existing in the search space can be preserved in the representation space. The Gray coding representation is part of a

family of bit representations that fall into this category (Whitley *et al.* 1998). We can convert any binary number to a Gray code number by performing XOR to its consecutive bits from right to left. For example, given the number 0101 in binary, we would do[1]: $1 \oplus 0 = 1$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, producing (the leftmost bit remains the same) 0111, which is the equivalent Gray code number.

The use of Gray coding has been empirically shown to improve the performance of a GA when applied to the classical De Jong test functions (DeJong 1975) (see for example Caruana and Schaffer 1988; Mathias and Whitley 1994*b*). In fact, Mathias and Whitley found that Gray coding not only eliminates Hamming cliffs, but also alters the number of local optima in the search space and the size of good search regions (those that will lead us to the vicinity of the global optimum). They showed that a random mutation hill-climber is able to find the global optimum of most of the test functions provided when Gray coding is used, even though some of these were designed to present difficulty to traditional (evolutionary or not) search algorithms.

3. Encoding real numbers

Although Gray coding can be very useful to encode integers, the problem of mapping the search space correctly onto the representation space becomes more serious when we try to encode real numbers. In the traditional approach (Wright 1991), a binary number is used to represent a real number, by defining lower and upper bounds for each variable, as well as the precision desired. For example, if we want to encode a variable that ranges from 0.35 to 1.40, using a 2-decimal precision, we would need $\log_2 (140–35) \approx 7$ bits to represent any real number within that range. However, in this case, we have the same problem that we previously discussed, because the number 0.38 would be represented as 0000011 whereas 0.39 would be encoded as 0000100[2].

Even with Gray coding there is another (more important) issue when dealing with real-world applications: high dimensionality. If we have too many variables, and we want high precision for each, then our binary strings will become extremely long, and the GA will tend to perform poorly.

We could adopt some standard binary format for representing real numbers such as the IEEE standard for single precision in which a real number is represented using 32 bits, from which 8 are used for the exponent in excess-127 notation and 23 bits are used for the mantissa (see Fig. 5.2) (Scragg 1992). We could handle a relatively large range of real numbers using a fixed amount of bits (for example between 2^{-176} and 2^{127} if we used the IEEE standard for single precision previously described). However, the decoding process would be more complex, and the mapping between the representation space and the search space would be much more complex than when a simple binary representation is used, because any small change in the

[1] $\oplus$ indicates XOR.
[2] We are assuming that 0.35 is encoded as 0000000.

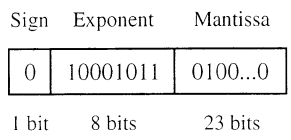

Fig. 5.2 An example of IEEE notation.

2.15	1.89	0.43	3.14	0.27	7.93	5.11

Fig. 5.3 An example of a real-coded GA.

exponent field would produce large jumps in the search space, whereas perturbations in the mantissa may not change, in a significant way, the numerical value encoded.

Whereas theoreticians claim that small alphabets should be more effective than large alphabets, practitioners have shown through a considerable number of real world applications (particularly numerical optimization problems) that the direct use of real numbers in a chromosome works better in practice than the traditional binary representation (Davis 1991; Eshelman and Schaffer 1993).

The use of real numbers in a chromosomic string (Fig. 5.3) has been common in other evolutionary techniques, such as evolution strategies (Schwefel 1981) and evolutionary programming (Fogel and Stayton 1994), where mutation is normally the primary operator. However, when dealing with GAs, there has been strong criticism of the use of real values in a chromosome, mainly because this higher cardinality representation makes behaviour of the GA more difficult to predict. Consequently, several special operators have been designed to emulate the effect of crossover and mutation over binary alphabets (Eshelman and Schaffer 1993; Wright 1991; Deb and Agrawal 1995).

Practitioners argue that one of the main abilities of real-coded GAs is their capacity to exploit the gradualness of functions of continuous variables (where gradualness is taken to mean that small changes in the variables correspond to small changes in the function). Real-coded GAs can thus adequately deal with the 'cliffs' produced when the variables used are real numbers, because a small change in the representation is mapped as a small change in the search space (Eshelman and Schaffer 1993; Wright 1991).

To reduce the gap between theory and practice, some researchers have developed a theoretical framework that justifies the use of higher-cardinality alphabets (Goldberg 1990; Wright 1991; Eshelman and Scaffer 1993; Surry and Radcliffe 1997), but there has been little agreement on most of the main issues, and the use of real-coded GAs remains a practitioner's choice.

Other representations for real numbers have been used. For example, the use of integers to represent each digit of a real number has been successfully

applied to several optimization problems (Coello *et al.* 1997*b*; Coello *et al.* 1998; Coello and Christiansen 1998). Fig. 5.4 shows a representation of the real number 1.45679. In this case, a fixed position is assumed for the decimal point in each variable, but this need not remain fixed for the other variables encoded in the string. Precision is limited by the length of the string, and can be increased or decreased as desired. The traditional crossover operators (one-point, two point, and uniform) can be used directly with this representation, and mutation may consist of generating a random digit for a certain location or of producing a small perturbation (for example ±1) to avoid large jumps in the search space. This representation is intended to be more of a compromise between a real-coded GA and a binary representation of real numbers, trying to keep the best from both worlds by incrementing the cardinality of the alphabet used, whilst leaving the use of the traditional genetic operators almost unchanged.

1	4	5	6	7	9

Fig. 5.4 An integer representation of real-numbers. The whole string is decoded as a single real number by multiplying and dividing each digit according to its location.

Alternatively, we could also use long integers to represent real numbers (Fig. 5.5) but then the operators would be redefined in the same manner as when using real numbers. The use of such a representation scheme seems unlikely as a replacement for real-coded GAs, because precision would be sacrificed and the only savings would be in terms of computer memory (the storage of integers takes up less memory than the storage of real numbers), but this has nevertheless been used in some applications (Davis 1991).

4. Variable-length representations

In some problems, the use of high-cardinality alphabets may be inappropriate, and it may be necessary to introduce variable-length chromosomes to deal with changes in the environment over time (for example, to decrease/increase the precision of a variable or to add/subtract variables). It may be possible to introduce symbols in the alphabet that are counted as 'empty' positions along the string, therefore allowing the use of fixed-length chromosomes to represent variable-length strings. This is the approach taken in (Coello *et al.* 1997*a*) to design combinational circuits. In this case, the use of a symbol called WIRE represents the absence of a gate, thereby allowing a change in the length of the resulting Boolean expression generated using a bi-dimensional matrix.

However, in other domains, this sort of simplification may be impossible, and alternative representations must be devised. For example, in problems that have either partial or full deception (Grefenstette 1993) (i.e., low-order building blocks do not guide the GA to the optimum and do not combine to

145679	67893	37568	95432

Fig. 5.5 Another integer representation of real-numbers. In this case, each gene contains an entire real number represented as a long integer.

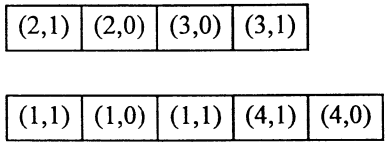

Fig. 5.6 Two examples of valid strings in a messy genetic algorithm.

form higher-order building blocks), a GA will not perform well regardless of its parameters. To deal with this problem, Goldberg *et al.* (1989, 1990, 1991) proposed a GA structure of variable length which uses populations of variable size, termed the 'messy GA' (mGA) as an alternative to the standard fixed-length, fixed population-size GA (Mitchell 1996).

MGAs start with short chromosomes, identify a set of good building blocks, and then increment the length of the chromosome to propagate these good building blocks along the rest of the string.

The representation used by mGAs is novel, since each bit is associated with a particular location along the string, and some locations could be assigned to more than one bit (overspecification) while others may not be assigned at all (underspecification). Consider, for example, the two strings shown in Fig. 5.6 which constitute valid chromosomes for a messy GA (we are assuming chromosomes of 4 bits). The notation adopted in this example uses parentheses to indicate a gene, which is defined as a pair consisting of the location along the string (the first value) and the bit value in that location (a binary alphabet is assumed). In the first case, the first and fourth positions are not specified, and the second and third are specified twice. In the second case, the second and third positions are not specified, the first is specified three times and the fourth is specified twice. To deal with overspecification, some simple deterministic rules may be defined. For example, we can use only the first definition from left to right for a certain location. For underspecification, we have to do something more complicated, because an underspecified string is actually representing a 'candidate schema' rather than a complete chromosome. For example, the first string in Fig. 5.6 represents the schema *10* (the * means 'don't care'). To compute fitness for an underspecified string, we can use a hill-climber to find a local optimum and then use that information to replace the 'don't cares' from the schema. This approach is termed 'competitive templates' by Goldberg *et al.* (1990).

Messy GAs operate in 2 phases (Goldberg *et al.* 1990): the 'primordial phase' and the 'juxtapositional phase'. In the primordial phase, short schemata are generated to serve as the building blocks of the juxtapositional phase in which they will be combined. The problem at this point is how to decide how long these 'short' schemata should be. If they are too short, they may not contain enough genetic material to solve the problem at hand; if they are too long, the technique will become impractical because of the 'curse of dimensionality' (we would have to generate and evaluate too many chromosomes).

During the primordial phase we generate these short schemata and evaluate their fitnesses. Subsequently, only selection is applied to the population

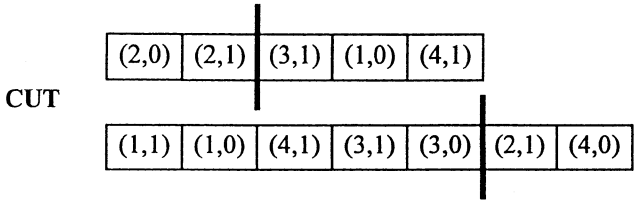

Fig. 5.7 An example of the 'cut' operator in a messy genetic algorithm. The thick lines indicate the cut point.

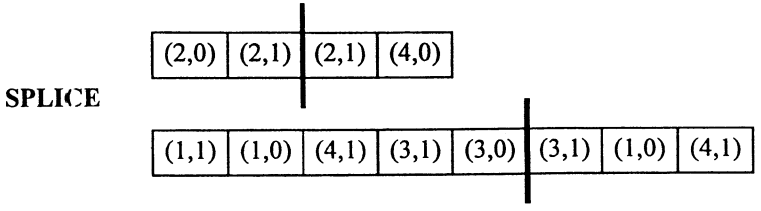

Fig. 5.8 An example of the 'splice' operator in a messy genetic algorithm. The thick lines show the part of the string that was added.

(without crossover or mutation) to propagate the good building blocks, and half of the population is deleted at regular intervals (Mitchell 1996). After a certain (pre-defined) number of generations, the primordial phase ends and the juxtapositional phase is begun. From this point, the size of the population remains fixed, and we use selection and two special operators called 'cut' and 'splice' (Goldberg *et al.* 1989). The cut operator simply takes a portion of the chromosome away, whereas splice puts two portions together. Consider the examples shown in Fig. 5.7 and 5.8.

Because of the nature of the messy GA, the strings produced by the cut and splice operators will always be legal. If the building blocks produced in the primordial phase carry enough information, the messy GA can approach the global optimum even if the problem is deceptive (Goldberg *et al.* 1991).

Although very promising, the drawbacks of messy GAs (Mitchell 1996) have kept them from widespread use, and only a few applications have been reported in the literature (Chowdhury and Li 1996; Kajitani *et al.* 1997; Iba *et al.* 1997; Halhal *et al.* 1997; Beveridge 1998).

5. Tree representation

One of the early goals of Artificial Intelligence (AI) was the automatic generation of computer programs. For many years this goal seemed too ambitious since there is normally an exponential growth of the search space as we extend the domain of a program, and consequently, any technique will produce either invalid or very inefficient programs.

There are many examples of evolutionary computing techniques that attempt to deal with automatic programming, but notorious failures even in

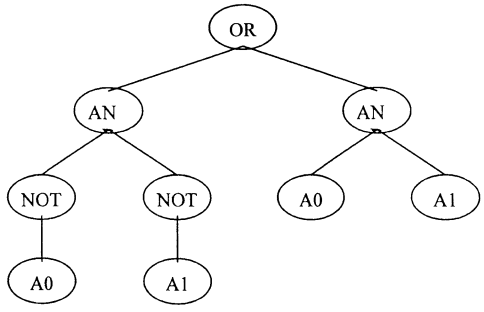

Fig. 5.9 An example of a chromosome in genetic programming.

very simple domains have prevented other AI researchers taking much of this work seriously (Fogel 1995). However, Holland developed the modern concept of the genetic algorithm within the framework of machine learning and much research still investigates the use of GAs for that purpose, although automatic programming was put aside by researchers for several years. One of the reasons for this was that a conventional GA has some (rather obvious) limitations when used for automatic programming, particularly in terms of representation issues. Encoding the set of instructions of a programming language and finding a way of combining them in a meaningful manner is not simple, but if a tree structure is used in combination with rules that avoid the generation of invalid expressions, we can build a primitive parser capable of producing simple programs. This was precisely the approach taken by John Koza (1992) to develop 'genetic programming' in which LISP was used to take advantage of the parser built into the language to evaluate the expressions produced.

The tree representation adopted by Koza requires different alphabets and specialized operators to evolve randomly generated programs until they become 100 per cent valid to solve a certain (pre-defined) task, but the underlying principles of the technique can be generalized. Trees are composed of functions and terminals. The functions normally used are the following:

1. Arithmetic operations (e.g., +, −, ×, ÷)
2. Mathematical functions (e.g., sine, cosine, log, exp)
3. Boolean operations (e.g., AND, OR, NOT)
4. Conditionals (IF-THEN-ELSE)
5. Iterators (DO-UNTIL)
6. Recursive functions
7. Any other function defined in the current domain

Terminals are typically variables or constants, and can be seen as functions that take no arguments. An example of a chromosome that uses the functions F = {AND, OR, NOT} and the terminals T = {A0, A1} is shown in Fig. 5.9.

Crossover can be performed by numbering the nodes of the trees corresponding to the 2 parents (Fig. 5.10) and selecting (randomly) a point in each so that the sub-trees below that point are exchanged (Fig. 5.11, where we assumed that the crossover point for the first parent is 2, and for the second is 6).

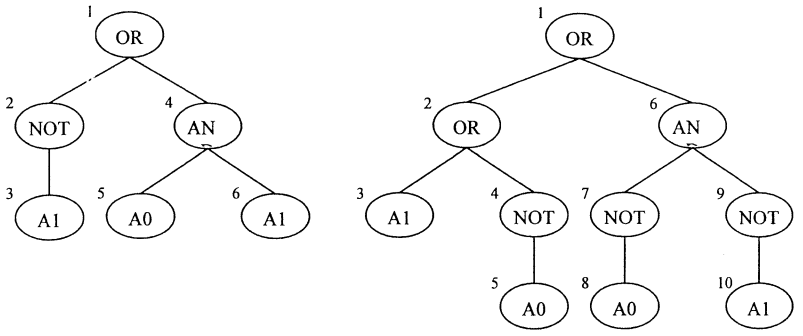

Fig. 5.10 Nodes in the tree are numbered as a previous step to crossover.

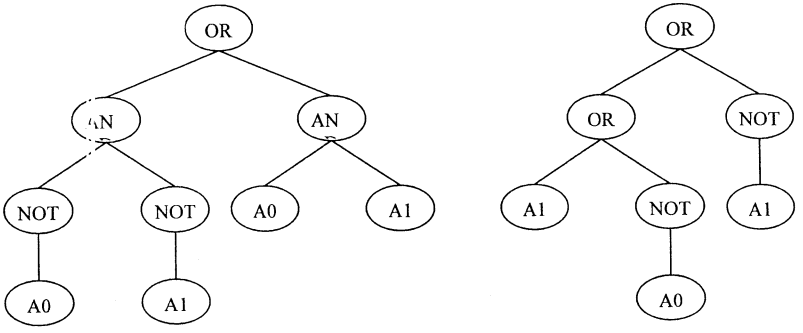

Fig. 5.11 The two children generated after performing crossover.

Typically, the sizes of the trees of the 2 parents will be different. It should also be observed that if the crossover point happens to be the root of one of the two parents, then that entire chromosome will become a sub-tree of the other parent, which is a way of incorporating subroutines into a program. It is also possible that the roots of both parents are selected as the crossover points. In that case, no crossover is performed, and the offspring are the same as their parents. Normally, the implementation of genetic programming imposes a limit to the maximum depth that a certain tree can reach, to avoid generating (randomly and by using crossover or mutation) trees of considerable size and complexity.

Mutation is performed by selecting (randomly) a certain point in a tree, and then replacing the sub-tree below it with another that is generated randomly. Fig. 5.12 shows an example in which the mutation point is 3.

Permutation is an asexual operator that emulates the inversion operator used in genetic algorithms (Goldberg 1989). It reorders the leaves of a sub-tree after a (randomly) selected point, aiming to strengthen the union of allele combinations with good performance in a chromosome (Holland 1975). An example of permutation is shown in Fig. 5.13, where the selected permutation point is 4 (the '*' indicates multiplication, and the '%' indicates 'protected division' and it refers to a division operator that avoids program crashes when the second argument is zero).

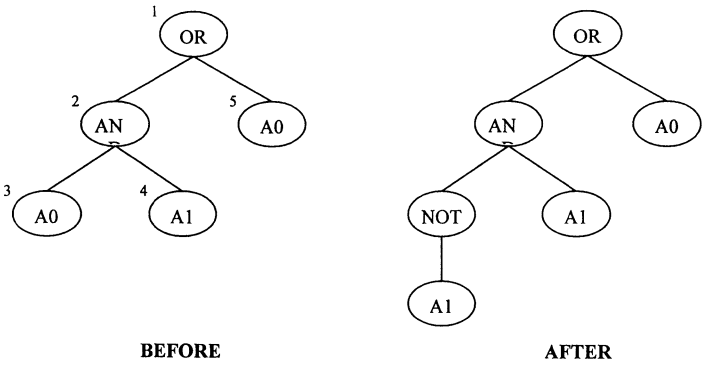

Fig. 5.12 An example of mutation in genetic programming.

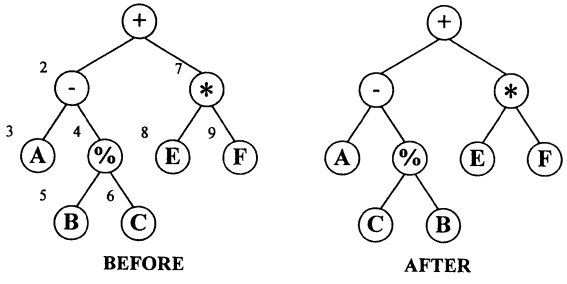

Fig. 5.13 An example of permutation in genetic programming.

It is also possible to protect or 'encapsulate' a sub-tree that is known to be a good building block, to avoid its destruction by any of the genetic operators. The selected sub-tree is replaced by a symbolic name pointing to the location of the sub-tree, and the actual sub-tree is compiled separately and linked to the rest of the tree in a way similar to external classes in an object-oriented language. Fig. 5.14 shows an example of encapsulation in which the right sub-tree is replaced by the name **EO**.

It is also normally necessary to do some editing to the expressions generated to simplify them, although the rules for doing that are problem-dependent. For example, if we are generating Boolean expressions, we can apply rules such as the following:

$$(AND\ X\ X) \rightarrow X$$
$$(OR\ X\ X) \rightarrow X$$
$$(NOT\ (NOT.\ X)) \rightarrow X$$

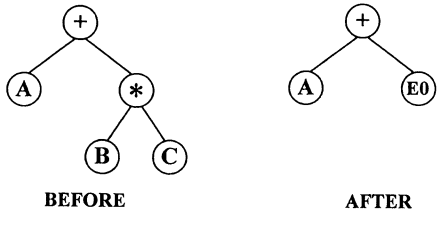

Fig. 5.14 An example of encapsulation in genetic programming.

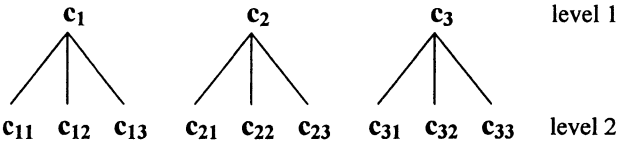

Fig. 5.15 An example of a 2-level structure of a Structured GA.

Finally, genetic programming provides mechanisms to destroy a percentage of the population so that we can refresh the chromosomic material after a number of generations. This mechanism, called execution, is useful in highly complex domains where the population may not contain a single feasible individual even after a large number of generations.

6. Structured genetic algorithm

Dasgupta (Dasgupta and McGregor 1994) proposed a representation that is a compromise between the traditional linear fixed-length chromosome and the tree encoding used by genetic programming. The *Structured Genetic Algorithm* (stGA) uses a hierarchical representation with a mechanism similar to diploidy (Goldberg 1989) where certain genes act as switching (or dominance) operators to turn genes on (active) or off (passive) respectively (Dasgupta and McGregor 1994).

The stGA uses a linear chromosomic string, but encodes a multi-level genetic structure (a directed graph or tree) as shown in Fig. 5.15. Genes at any level can be active or passive, but the high-level genes activate or deactivate sets of lower level genes, which means that any small change at a high level gets magnified at lower levels (Dasgupta and McGregor 1992). The idea is that the high-level genes should explore the potential areas of the space and lower-level sets should exploit that sub-space.

The hierarchical structure used by the stGA is nevertheless encoded as a fixed-length linear chromosome, as shown in Fig. 5.16. However, the data structure required to implement an stGA is slightly more complicated than the simple array required by a traditional GA, because each gene at a higher level acts as a switchable pointer which has two possible states: when the gene is active (on) it points to its lower level gene and when is passive (off), it points to the gene at the same level as itself (Dasgupta and McGregor 1992).

Other encodings have also been proposed. For example, Antonisse (1991) and Gero *et al.* (1994) have suggested the use of grammars in the context of programming languages and design, respectively. In fact, Antonisse claims that his approach is more general that Koza's (Koza 1992), because he defines context sensitive grammars which are more general (in the Noam Chomsky's hierarchy of languages) than the S-expressions used by Koza in

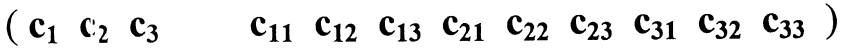

Fig. 5.16 A chromosome representing the 2-level structure of the structured GA shown before.

LISP (Antonisse 1991). Other (more problem-specific) encodings such as matrix representation have also been proposed (Vignaux and Michalewicz 1989; Beasley *et al.* 1993).

7. A case study

Parmee (1996) studied the application of the stGA in whole system design, utilizing the stGA to negotiate a design hierarchy described by both continuous and discrete variables. An example of this sort of hierarchical representation is shown in Fig. 5.17. In such a representation, differing sets of continuous variables are dependant upon discrete, selected configurations. The requirement therefore is for a search algorithm that can search across discrete design options, optimally sampling the many dependant continuous variable sets in order to identify high-performance design configurations (Parmee 1998).

Although some previous research on problems in which discrete and continuous design variables are interrelated has been reported (Jenkins 1991; Hajela and Lin 1992; Hajela and Lee 1995; Cai and Thierauf 1996), these approaches are sequential, normally tackling the discrete space first and then moving to the continuous space relevant to a certain configuration identified as optimum. The aim of Parmee's research was to explore concurrently both the discrete and the differing continuous search spaces. Previous experience with the stGA at the Plymouth Engineering Design Centre (Wade *et al.* 1994; Roberts and Wade 1994) led Parmee to use this approach as a starting point for the development of a suitable global search paradigm.

In his initial experiments, Parmee (1994, 1996) used binary representation for all the hierarchy. This presented problems due to the use of mixed discrete and continuous variables, resulting in encodings of different order, which produced, as a consequence, different probabilities of crossover and muta-

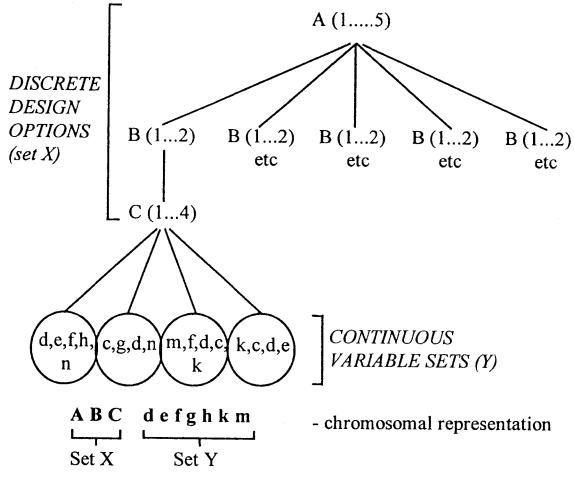

Fig. 5.17 A simple hierarchical representation of data.

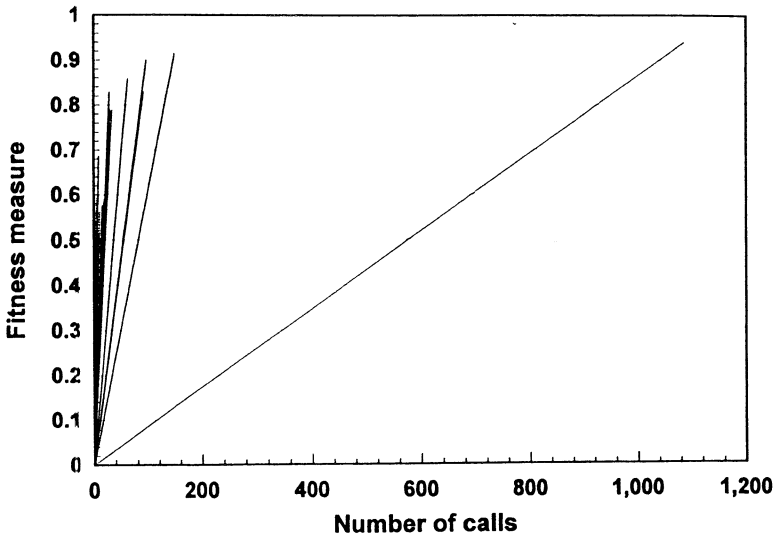

Fig. 5.18 Simple stGA implementation. Each line represents one of the possible 20 paths created by the discrete parameter set. The number of calls relates to the number of times each discrete set has been passed to the mathematical model. Fitness is relative to best fitness achieved during the experimentation.

tion for each variable, with higher order encodings having a much better chance of being disrupted than lower order encodings. A weighted scheme of the binary digits was introduced to deal with this problem. However, binary representation coupled with the inherent redundancy of variables within the stGA's structure resulted in very long chromosomes. The associated increase in chromosome length as the number of discrete levels increases therefore renders binary representation impractical for real-world applications.

A real number representation (Davis 1991) was therefore introduced to reduce such encoding problems. To encourage the GA to explore the lower-level variables in the hierarchy, an independent (higher) mutation probability was assigned to the high-level discrete variables, whilst maintaining a lower uniform mutation rate for the continuous variables (Fig. 5.19) (Parmee 1995). Variable mutation allowed greater exploration but resulted in premature convergence upon inferior solutions. This led Parmee to propose a hybrid approach, that would allow the stGA to conduct a diverse search across the hierarchy, while also identifying the best performing solutions. In initial experiments it was found found that the degree of search diversity across the hierarchy was very low (Fig. 5.18).

Although the stGA proved useful both in finding high performance solutions and exploring through a design hierarchy, the approach is inefficient for more complex hierarchies, because of the large number of parameters required as the hierarchy is developed and the complexity of the problem increases. Furthermore, because of the type of encoding, crossover tends to be disruptive at earlier generations even with a relatively simple hierarchy due to the exchange of information between differing configurations. This

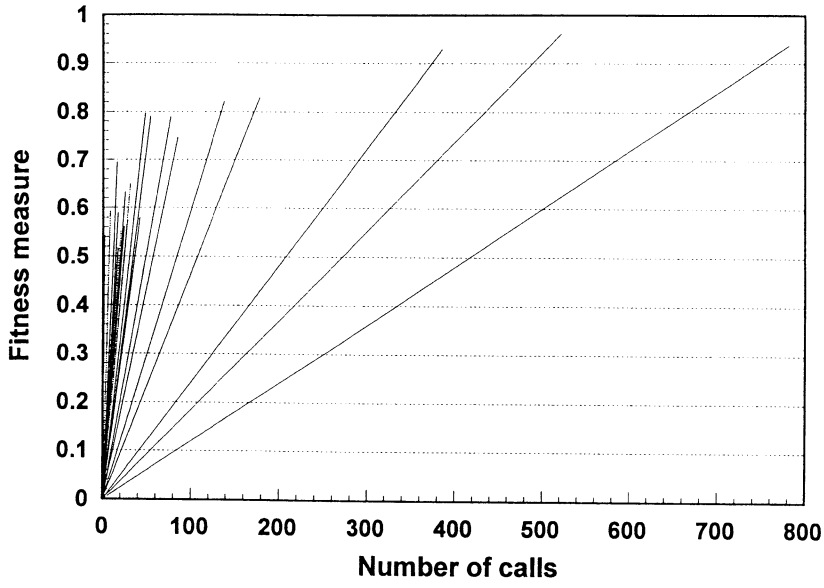

Fig. 5.19 Hybrid mutation regime.

exchange of information is largely random during early generations and generally results in premature convergence upon a locally optimal configuration.

Ideally, the representation would ensure the avoidance of non-feasible parameter combinations whilst allowing an appropriate information exchange through the traditional crossover operation. Parmee proposed a strategy that involved two individual search agents that operated simultaneously: a simple hill climber manipulating the discrete variables, and a GA manipulating the continuous variables. Selection and crossover allowed an inherent communication between the two sets of variables (discrete and continuous), and the approach was improved by introducing lower-level information exchange between the discrete variables by using elements of an ant colony metaphor for the manipulation of such discrete variables (Parmee 1998). This constitutes the basis for the GAANT algorithm that utilizes a genetic algorithm whilst also borrowing concepts from ant colony strategies (Parmee 1996; Parmee 1998).

7.1 GAANT

The GAANT algorithm borrows two concepts from the ant colony analogy (Coloni *et al.* 1991; Coloni *et al.* 1992; Bilchev and Parmee 1995):

- **Fitness proportionate distribution**: this is similar to fitness proportionate reproduction (Goldberg 1989), but in this case the amount of search resource (search agents) distributed along each discrete path is proportional to the relative strength (i.e., the fitness) of that combination of discrete options.
- **Evaporation**: the discrete path will be 'evaporated' if its strength does not improve over a preset number of cycles; the released search resource

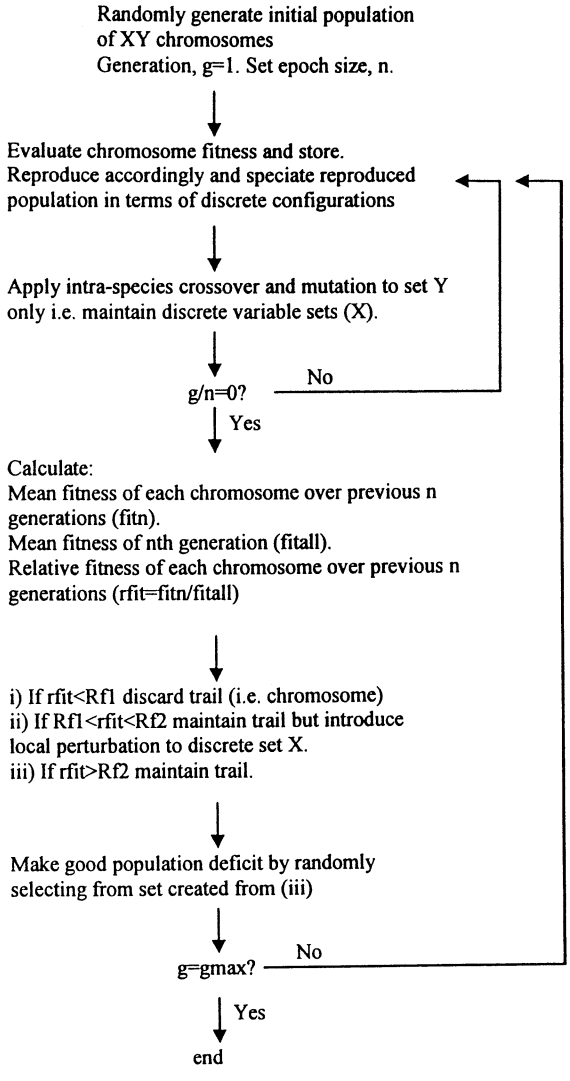

Fig. 5.20 GAANT flow chart.

(search agent) is then redistributed around the better sets of discrete design options within the hierarchy.

The concepts supporting these two operations were adapted slightly and integrated with the manipulation of the discrete variables by Parmee (1996*a*, 1997, 1998). Fig. 5.20 illustrates this integration. The values of the discrete variables are randomly selected at generation 1 and combined with a randomly selected population of continuous variables. The initial population of discrete values survives for a preset number of generations (n) while the associated continuous values are manipulated by a simple GA. A combination of like variable types during crossover and mutation is controlled by means of speciation of each chromosome in terms of like configurations of the discrete variables. Crossover then only occurs between members of the same species.

Table 5.1 Comparison of the stGA and the GAANT algorithm

	Hybrid stGA approach		GAANT implementation			
			$n = 5$		$n = 10$	
	2500 calls	5000 calls	2500 calls	5000 calls	5000 calls	7500 calls
Max solution	0.95	0.97	1	1.01	1	1.01
No. of solutions > 0.9	3	3	3	3	3	3
No. of solutions > 0.8	5	5	7	9	9	10
No. of calls along best path	611	1727	711	1277	1287	2545
SD of no. of calls (best path)	580	1706	207	460	413	738
SD of fitness (best path)	0.21	0.26	0.05	0.05	0.05	0.04
No. of missed paths	93	78	0	0	0	0
Max. no. of misses of any path	20	26	0	0	0	0

Evolution of the continuous variables continues over each generation whereas improvements in the discrete variables are achieved as follows: the average fitness of *each* chromosome is calculated over n generations, and then compared to the average fitness of the chromosomes of the nth generation. Evaporation, duplication, and perturbation of the discrete variables is then established in accordance with their relative fitness (**rfit**) which is represented in terms of their average fitness (**fitn**) over n generations and the average fitness of members of the nth generation (**fitall**) i.e., **rfit = fitn/fitall**. This allows the following communication:

- low-level communication between the chromosomes of the discrete set resulting in the evolution of the continuous variables within the bounds imposed by their discrete system configuration;
- low-level communication every nth generation between the chromosome sets representing the discrete variables which results in their gradual improvement;
- high-level communication between the two agents in the form of relative fitness of an entire string over n generations.

Evaporation, duplication, and perturbation are controlled at the nth generations by introducing two thresholds $\mathbf{Rf}_1$ and $\mathbf{Rf}_2$. If $\mathbf{Rf}_1 < \mathbf{Rf}_2$ then the configuration is evaporated (i.e., the chromosome is not reproduced). If **rfit** > $\mathbf{Rf}_2$ then the configuration is maintained (i.e., the chromosome is reproduced) and further resource is allocated from the evaporated configurations (i.e., the population deficit created by configuration evaporation is made good by randomly selecting chromosomes from those with a fitness higher than $\mathbf{Rf}_2$). Finally, if **rfit** lies between $\mathbf{Rf}_1$ and $\mathbf{Rf}_2$, the discrete variables are randomly perturbed to create a new configuration.

Table 5.1 compares the results found by Parmee (1998) using the dual mutation approach and GAANT and Fig. 5.21 shows the number of calls and fitness measure corresponding to the GAANT implementation. From these

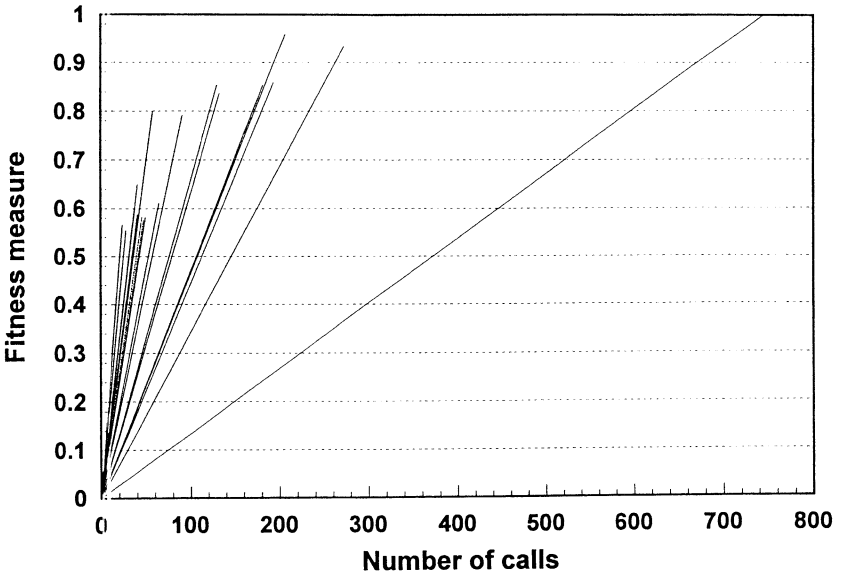

Fig. 5.21 GAANT implementation.

results it is clear that the GAANT approach provides improved performance in terms of maximum fitness across a larger number of paths than the stGA using a dual mutation strategy. The GAANT approach provides not only better results, but also covers the discrete paths better and is more robust than the stGA, as can be seen from the standard deviation (SD) over 50 runs of the algorithms.

GAANT has also been applied to more complex structures relating to the optimization of thermal power system configuration (Chen *et al.* 1997), obtaining a very significant reduction in design lead time in addition to significant increases in predicted power output. The integration of GAANT and some of its variants has replaced engineer/machine based design processing by a totally computer-based approach. Overall design time has been reduced by approximately 75 per cent (Parmee 1998).

7.2 Variable length hierarchies

The concepts behind the GAANT representation have also been applied to the manipulation of variable-length multi-level mathematical function representations. The objective in this case has been to improve the calibration of preliminary design models to empiric data or to results from a more in-depth analysis (FEA or CFD). This is achieved by identifying those areas of coding where insufficient knowledge or the requirement of keeping computational expense to a minimum has resulted in unavoidable function approximation. A contributing factor may be the inclusion of empirically derived coefficients. The objective is to evolve improved coding within these areas to achieve a better calibration with existing empiric data or results generated from a more in-depth computationally expensive analysis. If this is possible, the element

of associated risk would be lessened while rapid design iteration can still be achieved with these simple, but more representative models. This led to the use of genetic programming (Koza 1994) for system identification.

7.3 The need to improve genetic programming

Genetic Programming (GP) manipulation of engineering relationships has provided reasonable results related to the generation of formulas for pressure drop in turbulent pipe flow and also energy loss associated with sudden contraction or sudden expansion in incompressible pipeflow (Watson and Parmee 1996). However, it soon became apparent that the problems associated with the crossover of continuous coefficients between differing discrete functional structures was causing similar problems to those mentioned in relation to the design hierarchies of the previous section. The reason is that the exchange of information from continuous design spaces to unrelated discrete design configurations does not promote the formation of high performance variable parameter combinations.

Before describing how some elements of the GAANT strategy were introduced in the GP approach to improve its capabilities to deal with variable length design hierarchies, it is important to analyze the main limitations of GP.

In the past, at least two fundamental limitations of the traditional GP paradigm have been identified (Iba *et al.* 1995):

1. Random sub-tree crossover disrupts beneficial sub-trees in tree structures.
2. GP does not provide evaluation of tree descriptions.

Traditional GP blindly combines sub-trees by applying crossover operations. This can often disrupt beneficial sub-functions in tree structures. Thus, crossover operations seem ineffective as a means of constructing higher-order functions. Recombination operators (such as swapping sub-trees of nodes) often causes radical changes in the semantics of the trees. This *semantic disruption* (Iba *et al.* 1995) is due to the 'context-sensitive' representation of GP trees. As a result, useful sub-trees may not be able to contribute to higher fitness values of the whole tree, and the accumulation of useful sub-functions may be disturbed. To avoid this, Koza (1992, 1994) has proposed a strategy called *Automatic Defining Functions* (ADFs) for maintenance of useful sub-trees.

The fitness definitions used in traditional GP do not include evaluations of the tree descriptions. Without the necessary control mechanisms, trees may grow exponentially large, increasing the evaluation procedures, or so small that they degrade search efficiency. Usually the maximum depth of trees is set in order to control tree sizes, but an appropriate depth is not always known beforehand. Kinnear (1993) proposed using a size component in the fitness definition; i.e., the size of the tree is multiplied by a size factor, and the result is added to the raw fitness value. The use of a minimum description length (MDL) based fitness function for evaluating tree structures has been used together with a local hill-climber (Iba *et al.* 1995; Iba *et al.* 1993). This fitness definition involves a trade-off between certain structural details of the

tree and its fitting (or classification) of errors. In order to produce an efficient guided crossover operator to search the symbolic search space a symbolic function classification is required which can then be used to minimize *semantic disruption*. This classification, called Node Complexity weighting (NC) (Watson and Parmee 1997), includes information on the lengths of the individuals. Semantic disruption is therefore minimized while tree length is controlled.

Watson and Parmee (1997) proposed to combine steady state GP with NC controlled crossover using a technique called **RAM-***GP* (*Rapid, Attenuated Memory Genetic Programming*). In their approach, NC weightings are used as a basis for the crossover operator together with a high rate of mutation and steady GP. They extended the technique to incorporate sub-populations of solutions classified by the complexity of the root node of each individual. These sub-populations act as discrete GP sub-populations which communicate with each other via crossover. The new approach has been called **DRAM-GP** (**D** stands for *Distributed*).

The main concepts of DRAM-GP involve a steady state GP with constrained complexity crossover (CCC). Crossover is constrained by node complexity weighting values. The root node will give a complexity rating of the whole tree, and is thus used to speciate the population into smaller sub-populations. This approach was inspired by the speciation of like configurations relating to whole system design and the GAANT algorithm. These points are discussed in the remaining portion of this section.

Kinnear (1993) has investigated the use of steady state GP as an alternative to the generational model traditionally used (Koza 1992). The idea is to evaluate an individual immediately for fitness, and then merge it into the population (or in this case a species), in place of the existing lowest fitness individual. This approach is non-generational, but when the number of new individuals that have been generated is equal to the population size it is considered that the equivalent of a generation has passed. The population size is then the *total* number of individuals (i.e. species population size × number of species).

Node Complexity (NC) weighting is a measure of the complexity of a tree and all of its nodes. If for example we have a functional set and terminal set consisting of F = { +, –, *, %} and T = { a, b, c, d, e, f}, as in the two-box problem (Koza 1994), we can weight these functions (e.g. all terminals = 1.0, plus = 1.1, minus = 1.1, multiply = 1.2, divide = 1.2) each NC value is then a function of the NC values of the nodes below it and the weighting of that node. An example of the NC weighting is shown in Fig. 5.22.

Each node has a specific weighting factor which is applied to the NC values below them. The NC value is then the sum of these adjusted lower node values. It can be seen that the complexity of the tree will *decrease* with tree depth, for example in Fig. 5.22 NC[0] = 7.248 (the root node) and NC[7] = 1.0 (a terminal).

Crossover is then constrained by only crossing sub-trees with *similar* NC values (from initial runs using the weighting values given in the run parameters tables. All of the results shown here are restricted to between ± 2.0 NC). This then provides a numerical complexity measure which controls crossover

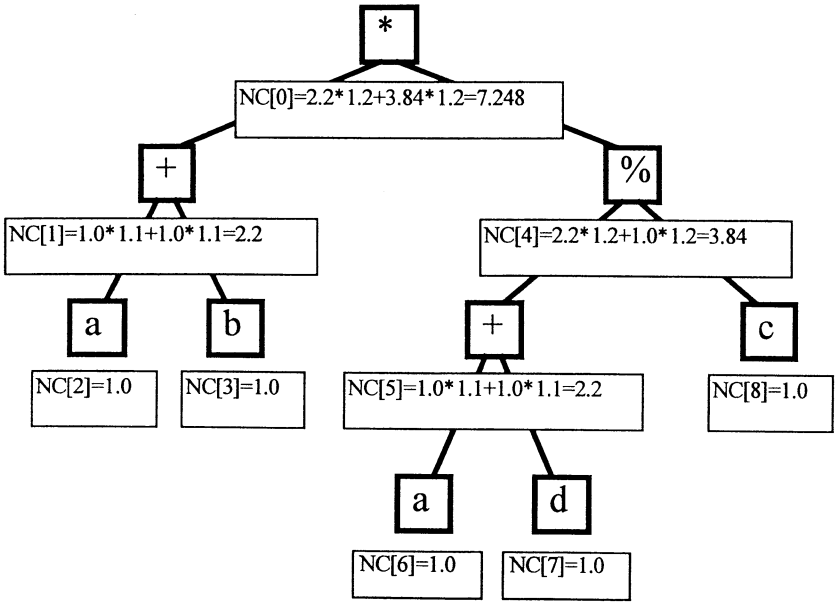

Fig. 5.22 Node complexity weighting.

and minimizes building block disruption by ensuring some similarity between crossed sub-trees. Tree lengths are also indirectly controlled.

The population is equally divided into sub-populations or 'species'. The run parameters that define the species groupings are the minimum and maximum NC values and the number of species used. The sub-populations are then divided equally between these two limits. If, for example, the minimum NC value is 10 and the maximum NC value is 40, with 3 species and a total population size of 300, then each species will have 100 individuals with the following NC[0] values:

Species 1: NC[0] > 10.0 and NC[0] < 20.0
Species 2: NC[0] > 20.0 and NC[0] < 30.0
Species 3: NC[0] > 30.0 and NC[0] < 40.0

Communication between sub-populations is achieved through crossover. As a new child individual is produced it is possible that its complexity changes, and if the root node complexity moves to another species range, it is placed into that species and evaluated. If a crossed individual's NC[0] value lies outside the species ranges then the individual is discarded.

Constrained Complexity Crossover (CCC) is initiated by randomly choosing parents P1 and P2 from the total population. A cross point CP1 is randomly chosen from P1 which then defines the root node of the sub-tree to be replaced. The second cross point CP2 from P2 *must* then be within ± 2.0 of the NC value of CP1. The sub-tree with root node CP2 replaces the sub-tree with root node CP1 with each allele having a probability of being mutated of FMUTATE (usually set to 0.5). When mutating, functionals can only be mutated to other functionals, and terminals into other terminals. Once

crossed, only one child is produced which is then evaluated and placed into the correct species population, replacing the worst individual within that species.

Injection mutation occurs every IM crosses (usually set to IM = Population Size) and changes *only one* allele within each individual with a probability of mutation FMUTATE (set to 0.5 throughout the work presented here). The top 5 individuals are elite and are *never* mutated, but are allowed to participate in crossover.

7.4 Boolean induction with DRAM-GP

Boolean concept learning (or Boolean induction) is an important part of machine learning, and can be regarded as a type of pattern recognition, in which the input (independent) and output (dependent) variables are binary. The effectiveness of DRAM-GP is initially demonstrated through one experiment. All calculations within this section are based on 100 runs. The results describe computational effort required to obtain one correct solution with a probability of 99 per cent. Computational effort E, and other performance calculations are discussed in Koza (1992, 1994).

7.4.1 Parity 3 problem (Koza 1992, 1994)

To show the effectiveness of DRAM-GP as a Boolean concept learner, a simple known experiment, 'parity 3', in which the goal function is the even parity function f of 3 variables is utilized. f takes the value 1 if the 3 input variables $x_1, \ldots, x_3$ have even parity, i.e. an even number of them are 1. The DRAM-GP parameters are shown in Table 5.2. The NC weightings for the functionals, N_F were chosen based upon the number of outputs that are true for each functional. The AND functional has 1 of 4 true values, and is thus considered more complex than the OR function which has 3 of 4 true outputs. Initially, the fitness was calculated after Koza (1992), i.e. Fitness = Test points – hits. This leads to individuals with the same fitness values but vastly differing complexities. A solution with a fitness of 4.0 and a NC[0] value of 8.88 should be ranked above another individual with the same fitness but a higher complexity. It was for this reason that the fitness measure was adjusted to: Fitness = (Test Points – Hits) + 0.001NC[0]. This then allows individuals of the same fitness but lower complexity to be ranked above those with higher complexity values.

Table 5.2 lists the parameters used for the GP implementation, and Table 5.3 shows a comparison of GP with DRAM-GP.

7.4.2 Two-Box problem (Koza 1992)

The two-box problem concerns the identification of a relationship between six independent variables ($x_1, \ldots, x_6$), where this relationship relates to the difference y in the volumes of the first box whose length, width, and height are x_1, x_2, x_3 and the second box whose length, width, and height are x_4, x_5, x_6.

Table 5.2 Run parameters for parity 3

Functional set	F = {and, or, nand, nor}
Arguments	F_A = {2, 2, 1, 3}
NC functionals	N_F = {l.2, 1.1, 1.1, 1.2}
Terminal set	T = {d0, d1, d2}
NC terminals	N_T = {1, 1, 1}
Mutation rate	0.5
Imutation (IM)	M (popsize)
Test points (TP)	8
Fitness	(TP–Hits) + 0.01NC[0]
NC max.	130.0
NC min.	50.0
Elite	5
CCC	± 2.0 of NC value
Chromosome length	100
Max. generations	200

Table 5.3 Parity 3 results

Method	Population size M (popsize × species)	Effort E
Koza[2](STD)	4000	80 000
Koza[3](STD)	16 000	96 000
Koza[3](ADF)	16 000	64 000
DRAM–GP	10 (10 × 1)	14 060
DRAM–GP	30 (30 × 1)	15 840
DRAM–GP	50 (50 × 1)	15 750
DRAM–GP	50 (10 × 5)	13 600
DRAM–GP	100 (10 × 10)	12 900
DRAM–GP	100 (20 × 5)	8400
DRAM–GP	150 (10 × 15)	9900
DRAM–GP	200 (10 × 20)	11 600
DRAM–GP	200 (20 × 10)	10 000
DRAM–GP	300 (20 × 15)	8400
DRAM–GP	400 (20 × 20)	7600

Thus $y = (x_1\ x_2\ x_3) - (x_4\ x_5\ x_6)$. The goal of this symbolic regression (i.e., the identification of a mathematical expression, in symbolic form, that provides a good, best, or perfect fit between a given finite sampling of values of the independent variables and the associated values of the dependent variables) is to derive the above equation as a 'complete form' when given a set of N observations.

In this problem, where the raw fitness is a floating point number rather than an integer, there is no need to include the NC[0] weighting in the fitness calculation. The multiply and divide functions are considered more complex

Table 5.4 Run parameters for two–box problem

Functiona set	F = { + , – , * , % }
Arguments	F_A= {2, 2, 2, 2}
NC Functionals	N_F = {1.1, 1.1, 1.2, 1.2}
Terminal set	T = {x1, x2, x3, x4, x5, x6}
NC Terminals	N_T = {1, 1, 1, 1, 1, 1}
Mutation Rate	0.5
Imutation (IM)	80
Test points (TP)	10
Fitness	MSE
Max. NC	30.0
Min. NC	5.0
Elite	5
CCC	± 2.0 of NC value
Chromosome length	50
Max. generations	200

Table 5.5 Two–box problem results

	Population size M (popsize × species)	**Effort E**
Koza [2] (STD)	4000	1 176 000
Koza [4] (ADF)	4000	2 220 000
DRAM–GP	10 (10 × 1)	95 760
DRAM–GP	20 (10 × 2)	163 800
DRAM–GP	30 (30 × 1)	60 000
DRAM–GP	50 (50 × 1)	54 800
DRAM–GP	50 (25 × 2)	76 500
DRAM–GP	50 (10 × 5)	292 500
DRAM–GP	80 (80 × 1)	66 640
DRAM–GP	100 (50 × 2)	213 000
DRAM–GP	100 (20 × 5)	260 000
DRAM–GP	100 (10 × 10)	222 000
DRAM–GP	150 (30 × 5)	112 500
DRAM–GP	200 (100 × 2)	184 800
DRAM–GP	200 (40 × 5)	156 000
DRAM–GP	200 (20 × 10)	166 000
DRAM–GP	250 (50 × 5)	169 500
DRAM–GP	300 (30 × 10)	130 500
DRAM–GP	400 (40 × 10)	136 000
DRAM–GP	500 (50 × 10)	237 000

than the plus and minus functions and thus have higher N_F values. The fitness measure is the mean squared error (MSE) of all of the test points. The DRAM-GP parameters are shown in Table 5.4 and the comparison of results between standard GP and DRAM-GP is shown in Table 5.5.

Summary

In this chapter we have discussed several possible data representations in evolutionary computing, starting from the traditional fixed-length linear structure in which each location along the string is occupied by a binary number. Some possible alternatives to using binary numbers have been discussed, such as integers and real numbers. In certain applications a fixed-length string may not be appropriate, and Goldberg's messy genetic algorithm which allows variable-length strings may be used.

A linear structure is often not appropriate for tasks such as automatic programming, where a tree structure seems more suitable. An interesting compromise between a linear and a tree structure is the Structured Genetic Algorithm. However, the inability of the Structured Genetic Algorithm to efficiently search across complex hierarchies has led to the development of the GAANT algorithm that utilizes a genetic algorithm whilst also borrowing concepts from ant colony strategies. The concepts introduced in the GAANT have been used to improve genetic programming performance through the development of a new technique that is able to achieve the results produced with Automatic Defining Functions (ADFs), whilst utilizing small populations.

References

Antonisse, H.J. (1991). A grammar-based genetic algorithm. In G.E. Rawlins, ed., *Foundations of genetic algorithms*, pp. 193–204, Morgan Kaufmann Publishers, San Mateo, California.

Beasley, D., Bull, D.R. and Martin, R.R. (1993). Reducing epistasis in combinatorial problems by expansive coding. In S. Forrest, ed., *Proceedings of the fifth international conference on genetic algorithms*, pp. 400–7, University of Illinois at Urbana-Champaign, Morgan Kaufmann Publishers, San Mateo, California.

Beveridge, J.R. (1998). Optimal 2D model matching using a messy genetic algorithm. In *Proceedings of the fifteenth national conference on artificial intelligence*. AAAI Press, Madison, Wisconsin.

Bilchev, G. and Parmee, I.C. (1995). The ant colony algorithm for searching continuous design spaces. In T.C. Fogarty, ed., Evolutionary computing. AISB workshop. Selected papers, *Lecture Notes in Computer Science*, pp. 25–39, Springer-Verlag, Sheffield, U.K.

Cai, J. and Thierauf, G. (1996). Structural optimization of a steel transmission tower by using parallel evolution strategy. In I.C. Parmee, ed., *Adaptive computing in engineering design and control '96. Proceedings of the second international conference*, pp. 18–25. Plymouth Engineering Design Centre, University of Plymouth, Plymouth, UK.

Caruana, R. and Schaffer, J.D. (1988). Representation and hidden bias: gray vs. binary coding for genetic algorithms. In *Proceedings of the Fifth International Conference on Machine Learning*, pp. 132–61. Morgan Kaufmann Publishers, San Mateo, California.

Chen, K., Parmee, I.C., and Gane, C.R. (1997). Dual mutation strategies for mixed-integer optimisation in power station design. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 385–90. Indianapolis, Indiana.

Chowdhury, M. M. and Li, Y. (1996). Messy genetic algorithm based new learning method for fuzzy controllers. In *Proceedings of the IEEE international conference on industrial technology*, IEEE Service Center, Shangai, China, http://eeapp.elec.gla.ac.uk/~chy/publications.html.

Coello, C. A. C. and Christiansen, A.D. (1998). Two new GA-based methods for multiobjective optimization. *Civil Engineering and Environmental Systems*, **15**, 207–43.

Coello, C. A. C., Christiansen, A.D., and Aguirre, A.H. (1998). Using a new GA-based multiobjective optimization technique for the design of robot arms. *Robotica*, **16**, 401–14.

Coello, C. A. C., Christiansen, A.D., and Aguirre, A.H. (1997*a*). Automated design of combinational logic circuits using genetic algorithms. In D.G. Smith, N.C. Steele, and R.F. Albrecht, eds. *Proceedings of the international conference on artificial neural nets and genetic algorithms ICANNGA '97* pp. 335–8. University of East Anglia, Springer-Verlag, Norwich, England.

Coello, C. A. C., Hernández, F.S., and Farrera, F. A. (1997*b*). Optimal design of reinforced concrete beams using genetic algorithms. *Expert Systems with Applications: An International Journal*, **12**(1).

Coloni, A., Dorigo, M., and Maniezzo V. (1991). Distributed optimisation by ant colonies. In *Proceedings of european conference on artificial life*, MIT Press Cambridge, Mass. pp. 134–42.

Coloni, A., Dorigo, M., and Maniezzo, V. (1992). An investigation of some properties of the ant algorithm. In R. Männer and B. Manderick, eds, Parallel problem solving from nature 2nd workshop, *Lecture notes in computer science*, pp. 509–20. North-Holland Publishing Company, Amsterdam.

Dasgupta, D. and McGregor, D.R. (1992). Nonstationary function optimization using the structured genetic algorithm. In *Proceedings of parallel problem solving from nature (PPSN 2)*, pp. 145–54. Springer-Verlag, Brussels.

Dasgupta, D. and McGregor, D.R. (1994). A more biologically motivated genetic algorithm: the model and some results. *Cybernetics and Systems: An International Journal*, **25**(3), 447–69.

Davis, L., ed. (1991). *Handbook of genetic algorithms*. Van Nostrand Reinhold, New York, New York.

Deb, K. and Agrawal, R.B. (1995). Simulated binary crossover for continuous search space. *Complex Systems*, **9**, 115–48.

De Jong, K.A. (1975). Analysis of the behaviour of a class of Genetic Adaptive systems. Dissertation Abstracts International vol 36, 5140B, University Microfilms no 76–9381.

Eshelman, L.J. and Schaffer, J.D. (1993). Real-coded genetic algorithms and interval-schemata. In L.D. Whitley, ed., *Foundations of genetic algorithms 2*, pp. 187–202. Morgan Kaufmann Publishers, San Mateo, California.

Fogel, D.B. (1995). *Evolutionary computation. Toward a new philosophy of machine intelligence*. The Institute of Electrical and Electronic Engineers, New York.

Fogel, D.B. and Stayton, L.C. (1994). On the effectiveness of crossover in simulated evolutionary optimization. *BioSystems*, **32**, 171–82.

Gero, J.S., Louis, S.J., and Kundu, S. (1994). Evolutionary learning of novel grammars for design improvement, *AIEDAM* **8**(3), 83–94.

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley Publishing Co., Reading, Massachusetts.

Goldberg, D.E. (1990). *Real-coded genetic algorithms, virtual alphabets and blocking*, Technical Report 90001, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Goldberg, D.E., Deb. K., and Korb, B. (1990). Messy genetic algorithms revisited: Studies in mixed size and scale, *Complex Systems*, **4**, 415–44.

Goldberg, D.E., Deb. K., and Korb, B. (1991). Don't worry, be messy. In R.K. Belew and L.B. Booker, eds, *Proceedings of the fourth international conference on genetic algorithms*, pp. 24–30. University of California, San Diego, Morgan Kaufmann Publishers, San Mateo, California.

Goldberg, D.E., Korb, B., and Deb, K. (1989). Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, **3**, 493–530.

Grefenstette, J.J. (1993). Deception considered harmful. In L.D. Whitley, ed., *Foundations of genetic algorithms 2*, pp. 75–91. Morgan Kaufmann, San Mateo, California

Hajela, P. and Lee, E. (1995). Genetic algorithms in truss topological optimization. *Journal of Solids and Structures.*, **32**(22), 3341–57.

Hajela, P. and Lin, C.Y. (1992). Genetic search strategies in multicriterion optimal design. *Structural Optimization*, **4**, 99–107.

Halhal, D., Walters, G.A., Ouazar, D., and Savic, D.A. (1997). Water network rehabilitation with a structured messy genetic algorithm. *Journal of Water Resources Planning and Management*, ASCE, **123**(3).

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Harbor.

Hollstien, R.B. (1971). *Artificial genetic adaptation in computer control systems.* PhD thesis, University of Michigan, Ann Harbor, Michigan.

Iba, H., deGaris, H., and Sato, T. (1993). System identification using structured genetic algorithms. In S. Forrest, ed., *Proceedings of the fifth international conference on genetic algorithms*, pp. 279–86. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers, San Mateo, California.

Iba, H., deGaris, H., and Sato, T. (1995). A numerical approach to genetic programming for system identification. *Evolutionary Computation*, **3**(4), 417–52.

Iba, H., Iwata, M., and Higuchi, T. (1997). Gate-level evolvable hardware: empirical study and application. In D. Dasgupta and Z. Michalewicz, eds, *Evolutionary algorithms in engineering applications*, pp. 259–76. Springer-Verlag, Berlin.

Jenkins, W.M. (1991). Towards structural optimization via the genetic algorithm. *Computers and Structures*, **40**(5), 1321–7.

Jong, A. K. D. (1975). *An analysis of the behaviour of a class of genetic adaptive systems*, PhD thesis, University of Michigan.

Kajitani, I., Hoshino, T., Iwata, M., and Higuchi, T. (1997). Variable length chromosome GA for evolvable hardware. In *Proceedings of the 1996 IEEE international conference on evolutionary computation*, pp. 443–7. Nagoya University, IEEE Service Center, Nagoya, Japan.

Kinnear, K.E. (1993). Generality and difficulty in genetic programming: evolving a sort. In S. Forrest, ed., *Proceedings of the fifth international conference on genetic algorithms*, pp. 287–94. University of Illinois at Urbana-Champaign, Morgan Kaufmann Publishers, San Mateo, California.

Koza, J.R. (1992). *Genetic programming. On the programming of computers by means of natural selection*, The MIT Press, Cambridge, Mass.

Koza, J. R. (1994). *Genetic programming II: automatic discovery of reusable subprograms*. The MIT Press, Cambridge, Mass.

Mathias, K.E. and Whitley, L.D. (1994*a*). Changing representations during search: a comparative study of delta coding. *Evolutionary Computation*, **2**(3), 249–78.

Mathias, K.E. and Whitley, L.D. (1994*b*). Transforming the search space with Gray coding. In J.D. Schaffer, ed., *Proceedings of the IEEE international conference on evolutionary computation*, pp. 513–18. IEEE Service Center, Piscataway, New Jersey.

Michalewicz, Z. (1992). *Genetic algorithms + data structures = evolution programs*, Second edn. Springer-Verlag, Berlin.

Mitchell, M. (1996). *An introduction to genetic algorithms*. The MIT Press, Cambridge, Massachusetts.

Parmee, I.C. (1994). The integration of adaptive search techniques with current engineering design practice, In I. C. Parmee, ed., *Adaptive computing in engineering design and control '94*, pp. 1–13. Plymouth Engineering Design Centre, University of Plymouth, Plymouth, UK.

Parmee, I.C. (1995). Diverse evolutionary search for preliminary whole system design. In I.C. Parmee, ed., *Proceedings of the 4th international conference on AI in civil and structural engineering*. Civil-Comp Press, Cambridge University.

Parmee, I.C. (1996). Cluster-oriented genetic algorithms (COGAs) and multi-agent strategies (GAANT). In I.C. Parmee, ed., *Adaptive computing in engineering design and control '96. Proceedings of the second international conference*, pp. 128–33. Plymouth Engineering Design Centre, University of Plymouth, Plymouth, UK.

Parmee, I.C. (1996*a*). The development of a dual-agent strategy for efficient search across whole system engineering design hierarchies. In Hans-Michael Voig, Werner Ebeling, Ingo Rechenberg and Hans-Paul Schwefel (eds), *Parallel problem solving from nature — PPSN IV*, pp. 523–32. Springer-Verlag, Berlin.

Parmee, I.C. (1998). Evolutionary and adaptive strategies for efficient search across whole system engineering design hierarchies. *Artificial intelligence for engineering design, analysis and manufacturing (AIEDAM)*. Cambridge University Press, Cambridge, England.

Parmee, I.C., Vekeria, H., and Bilchev, G. (1997). The role of evolutionary and adaptive search during whole system, constrained and detailed design optimization. *Engineering Optimization*, **29**, 151–76.

Roberts, A. and Wade, G. (1994). Optimisation of finite wordlength filters using a genetic algorithm. In I.C. Parmee, ed., *Adaptive computing in engineering design and control '94*, pp. 37–43. Plymouth Engineering Design Centre, University of Plymouth, Plymouth, UK.

Schwefel, H.P. (1981). *Numerical optimization of computer models*. John Wiley and Sons.

Scragg, G.W. (1992). *Computer organization. A top-down approach*. McGraw-Hill, New York.

Surry, P.D. and Radcliffe, N.J. (1997). Real representations. In R.K. Belew and M.D. Vose, eds, *Foundations of genetic algorithms 4*, pp. 343–63.

Vignaux, G. and Michalewicz, Z. (1989). Genetic algorithms for the transportation problem. *Methodologies for Intelligent Systems*, **4**, 252–9.

Wade, G., Roberts, A., and Williams, G. (1994). Multiplier-less FIR filter design using a genetic algorithm, *IEE Proceedings — Vision and Signal Processing*, **141**(3), 175–80.

Watson, A.H. and Parmee, I.C. (1996). Systems identification using genetic programming. In I.C. Parmee, ed., *Adaptive computing in engineering design and control '96.*

Proceedings of the second international conference, pp. 248–55. Plymouth Engineering Design Centre, University of Plymouth, Plymouth, UK.

Watson, A.H. and Parmee, I.C. (1997). Steady state genetic programming with constrained complexity crossover using species sub-populations. In T. Bäck, ed., *Proceedings of the seventh international conference on genetic algorithms*, pp. 315–21.

Whitley, D., Rana, S. and Heckendorn, R. (1998). Representation issues in neighbourhood search and evolutionary algorithms. In D. Quagliarella, J. Périaux, C. Poloni and G. Winter, eds, *Genetic algorithms and evolution strategies in engineering and computer science. Recent advances and industrial applications*, Chapter 3, pp. 39–57. John Wiley and Sons, West Sussex, England.

Wright, A.H. (1991). Genetic algorithms for real parameter optimization, In G.J.E. Rawlins, ed., *Foundations of genetic algorithms*, pp. 205–18. Morgan Kaufmann Publishers, San Mateo, California.

6 Applications of artificial neural networks to the analysis of multivariate data

Royston Goodacre

1. Multivariate data

Multivariate data consist of the results of observations of many different characters (variables) for a number of individuals (objects) (Mark 1991; Martens and Næs 1989). Each variable may be regarded as constituting a different dimension, such that if there are n variables each object may be said to reside at a unique position in an abstract entity referred to as n-dimensional hyperspace. This hyperspace is necessarily difficult to visualize, and the underlying theme of multivariate analysis (MVA) is thus *simplification* (Chatfield and Collins 1980) or dimensionality reduction, which usually means that we want to summarize a large body of data by means of *relatively* few parameters, preferably the two or three which lend themselves to graphical display with minimal loss of information.

In spectroscopy, variables are usually represented by properties such as the absorbance at particular wavelengths. Spectral techniques which seem ideally suited to analysis by multivariate methods include those based on vibrational measurements such as infrared (IR) absorbance and Raman light scattering. Other hyper-dimensional measurements include gas and liquid chromatography, nuclear magnetic resonance (NMR), and mass spectrometry (MS).

Conventionally the reduction of the multivariate data generated by MS (Goodacre and Kell 1996; Gutteridge *et al.* 1985; Magee 1993), chromatography (MacFie *et al.* 1978), IR and other spectroscopic methods (Defernez and Wilson 1995; Martens and Næs 1989), and NMR (Kvalheim *et al.* 1985) have been carried out using principal components analysis (Causton 1987; Chatfield and Collins 1980; Everitt 1993; Jolliffe 1986). PCA is a well-known technique for reducing the dimensionality of multivariate data whilst preserving most of the variance, and whilst it does not take account of any groupings in the data, neither does it require that the populations be normally distributed, i.e. it is a non-parametric method. Moreover, PCA can be used to identify *correlations* amongst a set of variables and to transform the original set of variables to a new set of *uncorrelated* variables called principal components (PCs). The objective of PCA is to see if the first few PCs account for most (>90 per cent) of the variation in the original data. If they do reduce the number of dimensions required to display the observed relationships, then

the PCs can more easily be plotted and 'clusters' in the data visualized; moreover this technique can be used to detect outliers. The closely related discriminant function analysis (DFA; sometimes referred to as canonical variates analysis (CVA) is often used to separate the objects (samples) into groups on the basis of the retained PCs and the *a priori* knowledge of the appropriate number of groupings; this is achieved by minimizing the within-group variance and maximizing the between-group variance (MacFie *et al.* 1978; Manly 1994; Windig *et al.* 1983). Provided that the data set contains 'standards' (i.e. known things) it is evident that one can establish the closeness of any unknown samples to a standard, and thus effect the identification of the former, a technique termed 'operational fingerprinting' by Meuzelaar *et al.* (1982).

2. Supervised *versus* unsupervised learning

Such analyses fall into the category of 'unsupervised learning' (Fig. 6.1), in which the relevant multivariate algorithms seek 'clusters' in the data (Everitt 1993). This allows the investigator to group objects on the basis of their perceived closeness in n-dimensional hyperspace. These methods, although in some sense quantitative, are better seen as qualitative since their chief purpose is to *distinguish* objects or populations. More recently, a variety of related but much more powerful methods, most often referred to within the framework of chemometrics, have been applied to the 'supervised' analysis of multivariate data. In these methods, of which multiple linear regression (MLR), partial least squares regression (PLS), and principal components regression (PCR) are the most widely used, one seeks to relate the multivariate spectral inputs to the concentrations of target determinants, i.e. to generate

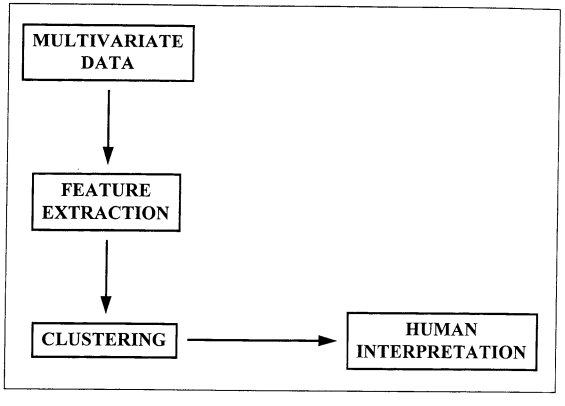

Fig. 6.1 Unsupervised learning. When learning is unsupervised, the system is shown a set of inputs (multivariate data) and then left to cluster them into groups. For multivariate analysis this optimization procedure is usually *simplification* or dimensionality reduction; this means that a large body of data (the inputs) are summarized by means of a few parameters with minimal loss of information. After clustering the results then have to be interpreted.

a quantitative analysis, essentially *via* suitable types of multidimensional curve fitting or regression analysis (Brereton 1992; Brown *et al.* 1996; Lavine 1998; Martens and Næs 1989; Massart *et al.* 1988). Although non-linear versions of these techniques are increasingly available (Berglund and Wold 1997; Heikka *et al.* 1997; Höskuldsson 1992; Taavitsainen and Korhonen 1992; Walczak and Massart 1996; Wold 1992; Wold *et al.* 1989), the usual implementations of these methods are linear in scope. However, a related approach to chemometrics, which is inherently nonlinear, is the use of (artificial) neural networks (ANNs).

For a given analytical system there are some patterns (e.g. mass spectra) which have desired responses which are known (i.e. the concentration of target determinands). These two types of data (the representation of the objects and their responses in the system) form pairs which for the present purpose are called inputs and targets. The goal of supervised learning is to find a *model* or *mapping* that will correctly associate the inputs with the targets (Fig. 6.2).

Thus the basic idea in these supervised learning techniques is that there are minimally 4 data sets to be studied, as follows. The 'training data' consist of (i) a matrix of s rows and n columns in which s is the number of objects and n the number of variables (these may be the absorbance at particular wavelengths, or the normalized ion intensities at a particular mass-to-charge ratio for MS, and (ii) a second matrix, again consisting of s rows and typically 1 or two columns, in which the columns represent the variable(s) whose values(s) are required (these are the result(s) wanted; Fig. 6.2) and which for the training set have actually been determined by some existing, 'benchmark' method. This variable may be the concentration of a target determinand, and is always paired with the patterns in the same row in (i). The 'test data' also consist of two matrices, (iii) and (iv), corresponding to those in (i) and (ii) above, but the test set contains different objects. As the name suggests, this second pair

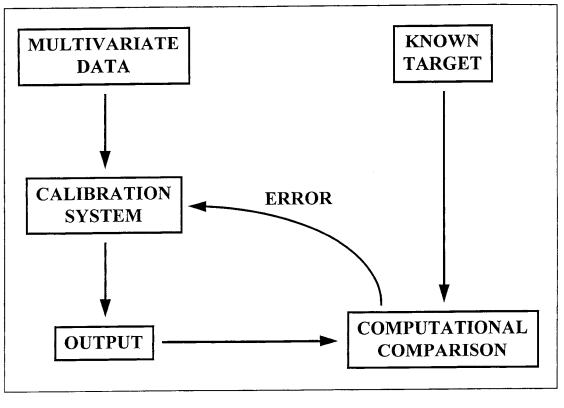

Fig. 6.2 Supervised learning. When we know the desired responses (targets) associated with each of the inputs (multivariate data) then the system may be supervised. The goal of supervised learning is to find a model that will correctly associate the inputs with the targets; this is usually achieved by minimizing the error between the known target and the model's response (output).

is used to test the accuracy of the system; alternatively they may be used to cross-validate the model. That is to say, after construction of the model using the training set (i, ii) the test data (iii) are 'passed' through the calibration model so as to obtain the model's prediction of results. These may then be compared with the known, expected responses (iv).

3. Good modelling practice

As in all other data analysis techniques, these supervised learning methods are not insensitive to badly chosen initial data (Kell and Sonnleitner 1995; Zupan and Gasteiger 1993). Therefore, the exemplars for the training set *must* be carefully chosen; the golden rule is 'garbage in — garbage out'. An excellent example of an unrepresentative training set was discussed on the BBC television programme *Horizon*; a neural network was trained to attempt to distinguish tanks from trees. Pictures were taken of forest scenes lacking military hardware and of similar but perhaps less bucolic landscapes which also contained more-or-less camouflaged battle tanks. A neural network was trained with these input data and found to differentiate successfully between tanks and trees. However, when a new set of pictures was analysed by the network, it failed to detect the tanks. After further investigation, it was found that the first set of pictures containing tanks had been taken on a sunny day whilst those without tanks were obtained when it was overcast. The neural network had thus learned to recognize the weather! We conclude that the training and test sets must be carefully selected to contain representative exemplars encompassing the appropriate variance over all relevant properties for the problem at hand.

It is also known (Bishop 1995; Goodacre and Kell 1993; Goodacre *et al.* 1994*a*; Kell and Sonnleitner 1995; Martens and Næs 1989; Wasserman 1989) that supervised learning methods such as neural networks (and partial least squares) can over-fit data. For example, an over-trained neural network may learn perfectly the stimulus patterns it has seen but can not give accurate predictions for unseen stimuli, i.e. it is no longer able to generalize. For supervised learning methods accurately to learn and predict the concentrations of determinands in biological systems, or to identify new observations as being from something previously seen, the model must be calibrated to the correct point. The reality is that in extension to normal chemometric practices detailed above the data should be split into three sets: (1) data used to calibrate the model; (2) date employed to cross-validate the model; (3) spectra whose determinand concentration, or identities, were 'unknown' and used to test the 'calibrated' system. During calibration, the models would be interrogated with both the training and the cross validation set and the error between the output seen and that expected calculated, thus allowing two calibration curves for the training and cross-validation sets to be drawn. When the error on the cross-validation data was lowest the system will be deemed to have reached the best generalization point and then may be challenged with input stimuli whose determinand concentrations, or identities, are really 'unknown'.

For quantitative determinations it is also imperative that the objects fill the sample space. If a neural network is trained with samples in the concentration range from 0 to 50 per cent it is unlikely to give accurate estimates for samples whose concentrations are greater than 50 per cent, that is to say, the network is unable to *extrapolate* (Kell and Sonnleitner 1995). Furthermore for the network to provide *good interpolation* it needs to be trained with a number of samples covering the desired concentration range (Goodacre *et al.* 1993*a*).

4. Applications of artificial neural networks

In the 1950s and 1960s researchers produced the first ANNs. Initially electronic circuits were used and these were later replaced by computer simulations. Then in 1969 Minsky and Papert (Minsky and Papert 1969) proved that single layer networks (perceptrons) were incapable of solving many simple problems, notably the function performed by the exclusive-or gate (XOR). This caused interest in ANNs to diminish rapidly and this science was put to sleep for 20 years. In time, several workers (see Parker 1982; Rumelhart *et al.* 1986) independently invented backpropagation ANNs which used a hidden layer employing a sigmoidal squashing function. These new ANNs were able to solve many of the problems posed by Minsky and Papert. This finding led to an explosion of interest and research into ANNs, both theoretical and in their application. So great has been the resurgence of interest into ANNs that since 1986 the number of publications has grown exponentially (Fig. 6.3) and in 1998 over 3000 papers have been published (not including conference proceedings).

Figure 6.3 also illustrates that there has been a wealth of papers on ANN applications totalling approximately 1000 in 1998, and that this has been a

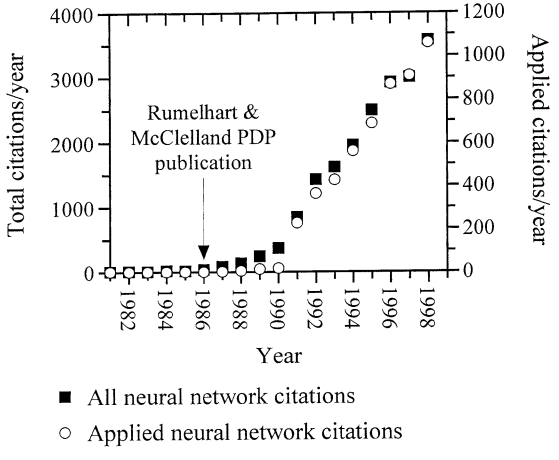

Fig. 6.3 Survey of papers on neural networks and references on neural networks applied to spectral techniques from 1981 to 1998. Source, BIDS Institute for Scientific Information Inc. Service (http://www.bids.ac.uk/isi.html/).

constant one third of all the papers, indicating the great interest globally in neural computational technology.

It would be impossible to review so many publications, therefore what follows are selected examples which fall into one of three domains; (1) exploratory analyses, (2) classification and identification, or (3) quantification.

Exploratory data analyses are those where little useful information is available on the problem in hand and we would like to know how similar a group of objects are. For example, in this set of five plastic samples are any the same and which are different? These seven bottles of olive oil are all labelled identically — but is this true? Or given a diverse population of bacteria, which are similar and which different? Exploratory data analysis is also used to detect outliers, the acquisition of a spectral measurement may be complicated and prone to experimental error; by analysing several replicates one can find out whether all spectra are identical or whether there has been a problem (perhaps a fly was caught in the analysis). Outlier detection may also include looking for novelty, for example my microbial culture collection contains several thousand isolates producing potentially novel pharmaceuticals; when cultivating 'new' bacteria how do I know that I have not previously characterized this organism?

There is much overlap between classification and identification, and it is usual to deal with these two together since classification is the discipline whose aim is to identify objects. For example, from which polymer is a particular plastic made? Where was this foodstuff produced? Is this bacterium isolated from a patient the same pathogen isolated from another patient in ward X?

Quantification problems are those where the aim is to predict the amount of a substance. For example, how much of the co-polymers poly-hydroxybutyrate and poly-hydroxyvalerate are in this biodegradable plastic? In the orange juice I have bought how much is from Brazil and how much from Florida? Or with respect to my patient in ward X that has bacterial septicaemia what is the microbial load in his or her blood?

5. Exploratory data analyses

Exploratory data analyses fall into the category of 'unsupervised learning', in which the relevant multivariate algorithms seek 'clusters' in the data (Everitt 1993); the most common multivariate statistical method is principal components analysis (PCA). Recently there has been an interest in the use of neural computation methods which can also perform unsupervised learning on multivariate data, the most commonly used are self-organizing (feature) maps (SOMs) and auto-associative artificial neural networks.

5.1 Self-organizing maps

SOMs were invented by Teuvo Kohonen (Kohonen 1989), and are hence also referred to as Kohonen ANNs. SOMs provide an objective way of classifying data through self-organizing networks of artificial neurons (Hecht-Nielsen 1990; Hertz *et al.* 1991; Kohonen 1989).

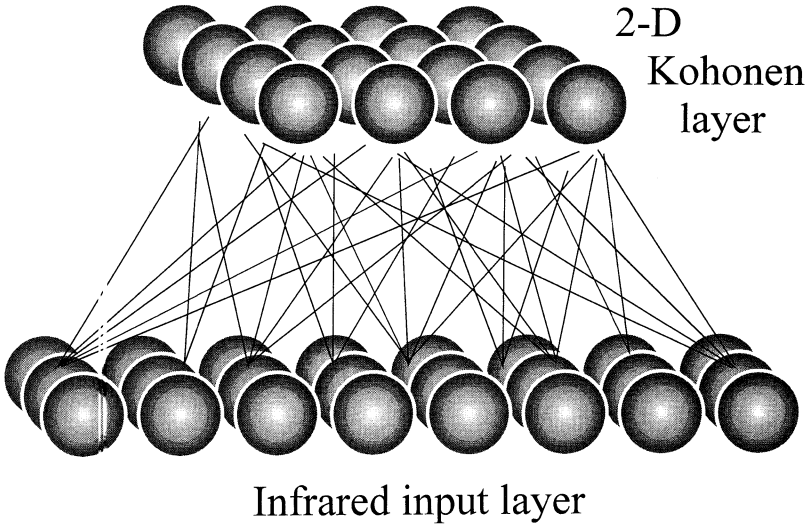

Fig. 6.4 A simplified self-organizing map. Nodes in the two-dimensional Kohonen layer are interconnected with each other, such that an activation node tends to activate surrounding nodes also. The infrared data are applied to the input layer (represent here by only 24 nodes; in reality >800 inputs) which activates a node or group of neighbouring nodes in the Kohonen layer (represented here as having 4×4 nodes; the number of nodes can be varied to allow quantitative information to be extracted).

SOMs used to analyse spectral data typically consist of a two-dimensional network of neurons arranged on a square grid (Fig. 6.4). Each neuron is connected to its eight nearest neighbours on the grid. The neurons store a set of weights (a weight vector) each of which corresponds to one of the inputs in the data. Thus, for infrared data consisting of 882 quantitative absorbances (see Fig. 6.5 for examples of IR spectra) at particular electromagnetic radiation wavelengths each node stores 882 weights in its weight vector. Upon presentation of an infrared spectrum (represented as a vector consisting of the 882 absorbances) to the network each neuron calculates its 'activation level'. A node's activation level is defined as:

$$\sqrt{\sum_{i=0}^{n} (weight_i - input_i)^2}$$

This is the Euclidean distance between the points represented by the weight vector and the input vector in n-dimensional space. Thus a node whose weight vector closely matches the input vector will have a small activation level, and a node whose weight vector is very different from the input vector will have a large activation level. The node in the network with the smallest activation level is deemed to be the 'winner' for the current input vector.

During training the network is presented with each input pattern in turn, and all nodes calculate their activation levels. The winning node and some of the nodes around it are allowed to adjust their weight vectors to match the

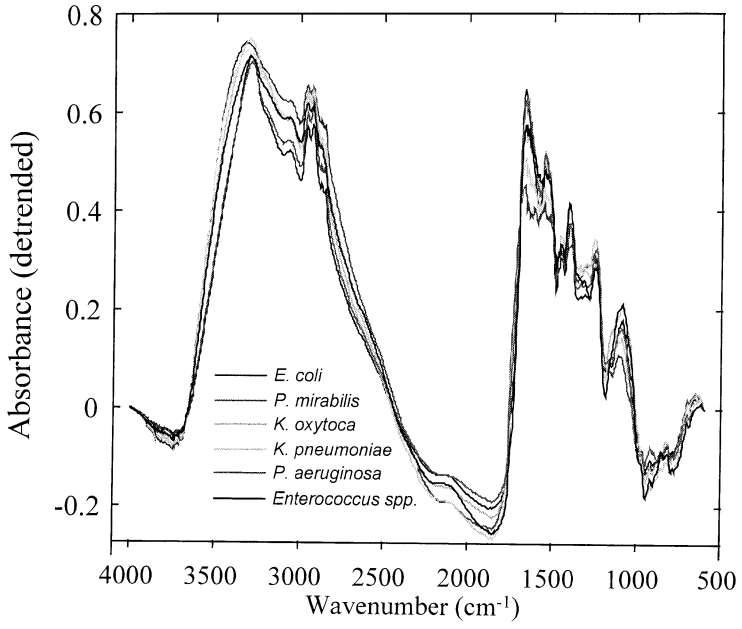

Fig. 6.5 Fourier transform infrared diffuse reflectance–absorbance spectra of bacteria typically associated with urinary tract infection; *Escherichia coli, Proteus mirabilis, Klebsiella oxytoca, Klebsiella pneumoniae, Pseudomonas aerginosa,* and an *Enterococcus* species.

current input vector more closely. The nodes in this set are said to belong to the 'neighbourhood' of the winner. The size of the winner's neighbourhood is varied through training. Initially all nodes in the network are included in the neighbourhood of the winner, but as training proceeds the size of the neighbourhood is decreased linearly after each presentation of the complete 'training set' (all the spectra being analysed), until it includes only the winner itself. The amount by which the nodes in the neighbourhood are allowed to adjust their weights is also reduced linearly throughout the training period.

The factor which governs the size of the weight alterations is known as the learning rate and is represented by α. The iterative adjustments to each item in the weight vector (where δw is the change in the weight) are made in accordance with the following:

$$\delta w_i = -\alpha(w_i - i_i)$$

This is carried out for $i = 1$ to $i = n$ where in this case $n = 882$. The initial value for α is 1 and the final value is 0.

The effect of the 'learning rule' (weight update algorithm) is to distribute the neurons evenly throughout the region of n-dimensional space populated by the training set (Hecht-Nielsen 1990; Hertz *et al.* 1991; Kohonen 1989).

This effect is displayed in Fig. 6.6 which shows the distribution of a square network over an evenly populated two-dimensional square input space (Fig. 6.6(A)), and a more complex input space (Fig. 6.6(B)). The neuron with the weight vector closest to a given input pattern will win for that pattern and

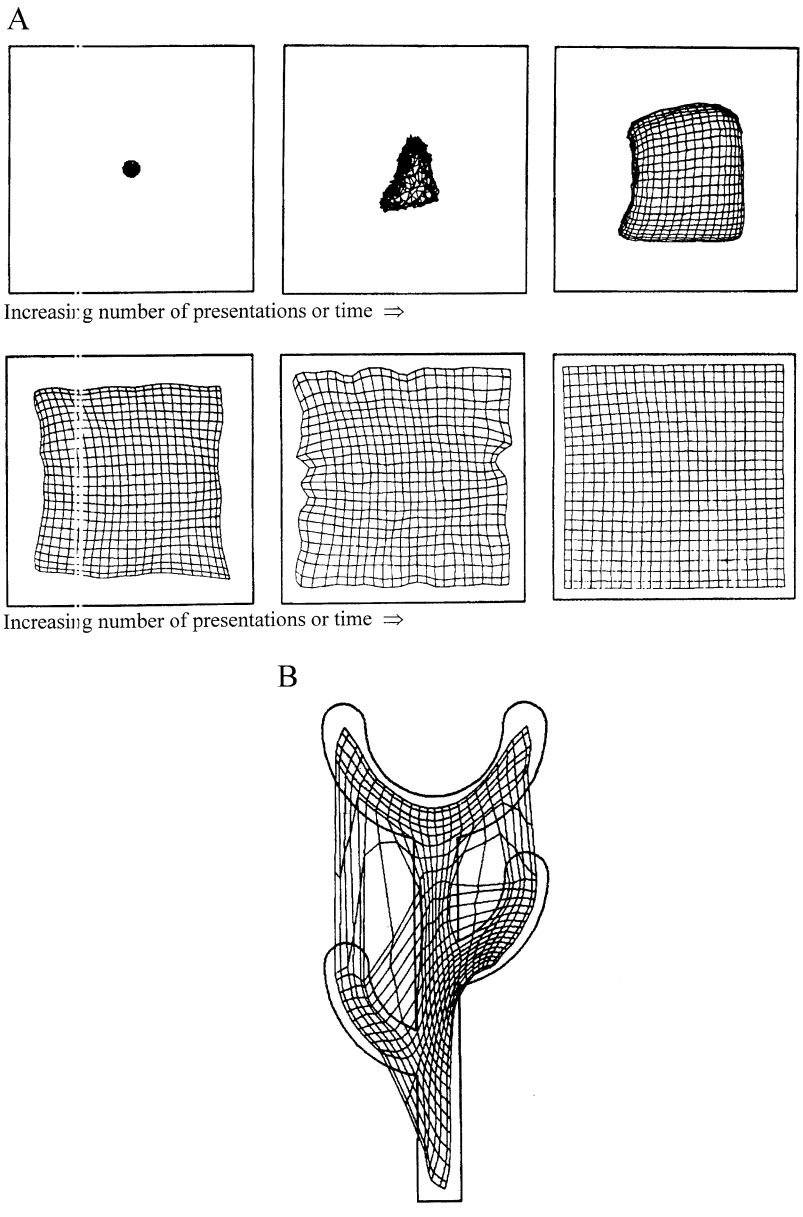

Fig. 6.6 Representations of square networks distributed across (A) an evenly distributed square, and (B) a more complex two-dimensional input space.

for any other input patterns that it is closest to. Input patterns which allow the same node to win are then deemed to be in the same group, and when a map of their relationship is drawn a line encloses them. By training with networks of increasing size a map with several levels of groups or 'contours' can be drawn. These contours, however, may sometimes cross, which appears to be due to failure of the SOM to converge to an even distribution of neurons over the input space (Erwin *et al.* 1992).

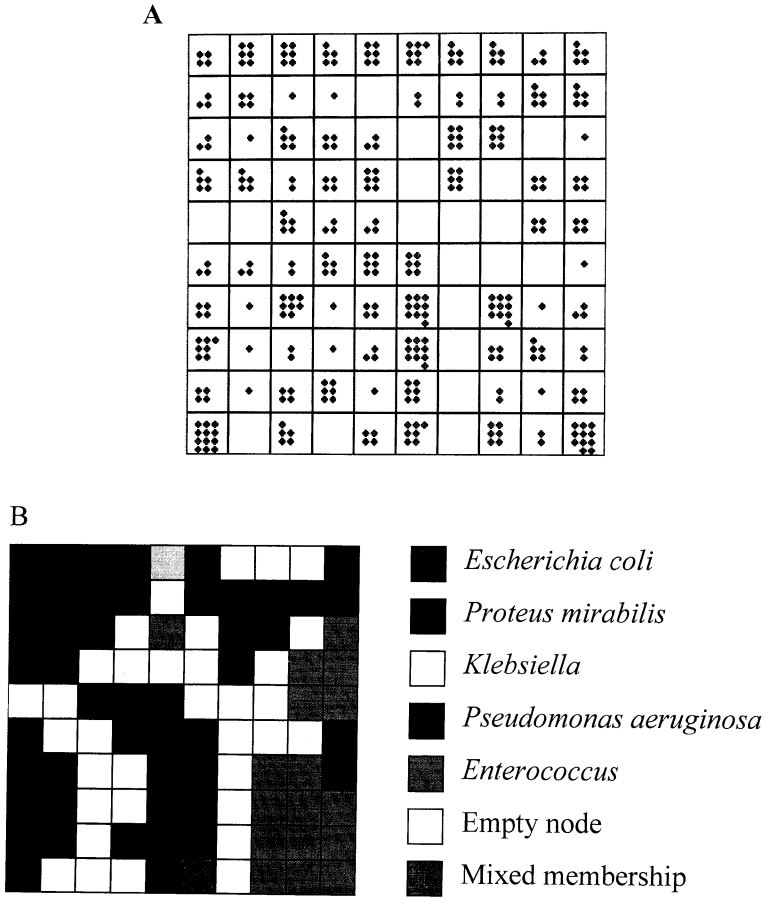

Fig. 6.7 Kohonen map (10 by 10 square grid) from a SOM trained with infrared data from measurements made on bacteria. Two maps are show: (A) the number of spectra that are found in each node in the 2-dimensional Kohonen layer and (B) the identities of the bacteria

Construction of these maps allows close examination of the relationships between the items in the training set, which in this example consisted of infrared spectra derived, for example, from bacterial species (Fig. 6.7).

Networks on square grids of 10 nodes were used to group the IR spectra. The SOMs were allowed to 'wrap around' so that they formed toroidal structures to avoid the edge effects which otherwise tend to corrupt very small networks of this type. The result of this exploratory analysis is displayed in Fig. 6.7 as a 2-dimensional Kohonen map. This groups the bacteria together; one can observe that the same bacteria cluster in the output nodes of this 10 by 10 grid.

Elsewhere, Wilkins *et al.* (Wilkins *et al.* 1994*a*, 1994*b*, 1996) have applied Kohonen maps to multi-dimensional flow cytometric data for the identification of species of fresh water phytoplankton, and Goodacre and colleagues have exploited SOMs to carry out unsupervised learning, and hence the classification of canine *Propionibacterium acnes* isolates (Goodacre *et al.*

1994*b*), *P. acnes* isolated from man (Goodacre *et al.* 1996*a*), and plant seeds (Goodacre *et al.* 1996*b*). SOMs have also been used to detect and classify human blood plasma lipoprotein lipid profiles on the basis of H^1NMR spectroscopic data (Kaartinen *et al.* 1998), for cluster analysis of multivariate satellite data (Waldemark 1997), and for seismological surveys of earthquakes and quarry blast (Musil and Plesinger 1996).

5.2 Auto-associative artificial neural networks

Auto-associative artificial neural networks (AAANNs) are a neural network-based method again used for unsupervised feature extraction and were pioneered by Kramer (Kramer 1991, 1992).

AAANNs consist of five layers containing processing nodes (neurons or units) made up of a layer of x input nodes (for example in Fig. 6.8 this is depicted as a set of 24 measurements), x output nodes (the same set of 24 measurements as used in the input layer), and three 'hidden' layers containing in this example of 8, 2 and 8 nodes respectively; this may be represented as a 24-8-2-8-24 architecture.

Adjacent layers of the network are fully interconnected, and the algorithm used to train these neural networks is the standard back-propagation (BP)

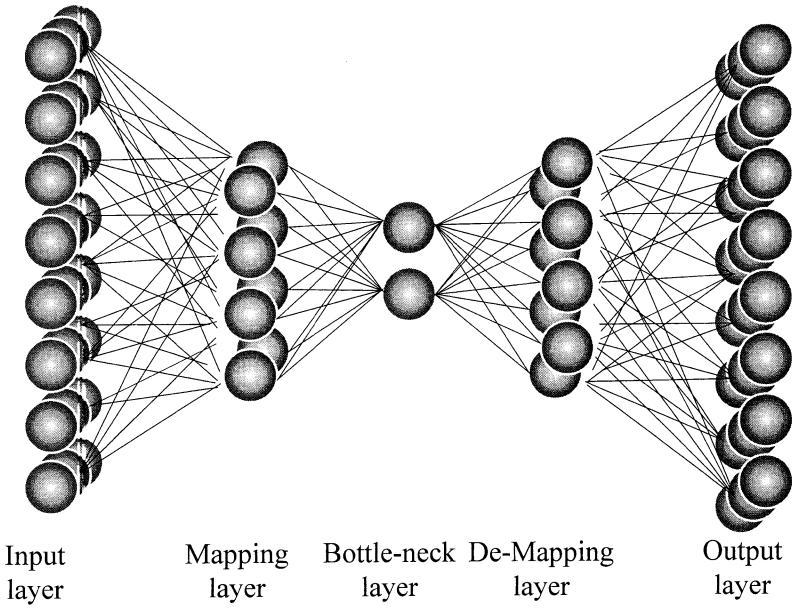

Fig. 6.8 Architecture of an auto-associative neural network consisting of 5 layers. In the architecture shown, adjacent layers of the network are fully interconnected. The input and output layer are presented with identical multivariate data (in this figure there are 24 nodes in these layers). A key feature of the auto-associative network is the data compression in the middle (third) bottle-neck layer of 2 nodes. The second and fourth layers each consisted of 8 nodes and these map and de-map the mass spectra allowing feature extraction in the bottle neck layer; this is equivalent to non-linear principal components analysis.

(Chauvin and Rumelhart 1995; Rumelhart *et al.* 1986; Werbos 1994). Since these neural networks are auto-associative in nature, that is to say, during training the input and output layer are presented with identical multivariate data, a key feature of these networks is the data compression in the middle (third) bottle-neck layer of 2 nodes. The second and fourth layers each consist of nodes that map and de-map the multivariate data, allowing feature extraction in the bottle neck layer. This is equivalent to non-linear principal components analysis (Kramer 1991, 1992). After training, each of the multivariate data used to train the AAANN is applied in turn to the input layer and the overall activation on the three nodes in the 'bottle-neck' layer calculated. Plots of the activations of the nodes in the 'bottle-neck' layer therefore allow 'clusters' to be found in the data.

Kramer and colleagues (Kramer 1992; Leonard and Kramer 1993) have shown that these types of neural networks reduce measurement noise by mapping inputs into the space of the correlation model (the cluster analyses in the middle bottleneck layer), and the residuals of this mapping can be used to detect sensor failures; moreover values for missing and faulty sensors can be estimated using these networks.

Auto-associative ANNs have been used to reduce the dimensionality of the infrared spectra of polysaccharides and hence extract spectral features due to polysaccharides (Jacobsson 1994), to detect plasmid instability using on-line measurements from an industrial fermentation producing a recombinant protein expressed by *Escherichia coli* (Montague and Morris 1994), for effecting exploratory cluster analyses of pyrolysis mass spectra (Goodacre *et al.* 1996*b,c*), and for knowledge extraction in chemical process control (Kuespert and McAvoy 1994). Whilst, more recently NLPCA using backpropagation ANN has been used for image coding (Tzovaras and Strintzis 1998) and image processing (Bowden *et al.* 1997), and for electrocardiogram (ECG) analysis for detecting ischemia in patients (Stamkopoulos *et al.* 1998).

6. Identification

The goal of identification is to take an unknown object and classify it as belonging to a group that has been seen previously. This process is based on supervised analysis and the two most exploited of the neural computational methods for this purpose are (1) multilayer perceptrons (MLPs) using standard backpropagation of error and (2) radial basis function neural networks (RBFs).

In MLPs and RBFs that are to be trained for identification purposes, as detailed above the training data used to calibrate the model consist of (i) a matrix of s rows and n columns in which s is the number of objects and n the number of variables, and (ii) a second matrix, again consisting of s rows and the same number of columns as there are classes to be identified. For identification these s rows are binary encoded as shown in Fig. 6.9; these are the result(s) wanted and which for the training set have actually been determined by classical identification methods, and are always paired with the pat-

	Nodes on MLP or RBF output layer						
	1	2	3	4	5	6	7
A	1	0	0	0	0	0	0
B	0	1	0	0	0	0	0
C	0	0	1	0	0	0	0
D	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0
F	0	0	0	0	0	1	0
G	0	0	0	0	0	0	1

Fig. 6.9 Binary encoding the seven nodes in the output layer on a multilayer perceptron or radial basis function trained to classify one of seven substances A–G.

terns in the same row in (i). Once trained, new input data can be passed through these ANNs, and the identities 'read off' easily, since a tabular format is employed in the classification encoding.

The following texts, books, and indeed this AI handbook are recommended as excellent introductory texts to artificial neural networks (Baxt 1995; Bishop 1995; Dybowski and Gant 1995; Goodacre *et al.* 1996*d*; Haykin 1994; Hertz *et al.* 1991; Richard and Lippmann 1991; Ripley 1994; Ripley 1996; Rumelhart *et al.* 1986; Simpson 1990; Wasserman 1989; Werbos 1994; Zupan and Gasteiger 1993). The following briefly outlines the major differences between MLPs and RBFs.

6.1 Multilayer perceptrons

The structure of a typical MLP is shown in Fig. 6.10(A). It consists of three layers; multivariate data as the input layer, connected to an output layer encoded for identification purposes as detailed in Fig. 6.9, via a single hidden layer. Each of the input nodes is connected to the nodes of the hidden layer using abstract interconnections (connections or synapses). These connections each have an associated real value, termed the weight (w_i), that scales the input (i_i) passing through them, this also includes the bias (ϑ), which also has a modifiable weight. Nodes sum the signals feeding to them (*Net*):

$$Net = i_1w_1 + i_2w_2 + i_3w_3 + \ldots + i_iw_i + \ldots + i_nw_n = \sum_{i=1}^{n} i_iw_i + \vartheta$$

The sum of the scaled inputs and the node's bias, is scaled to lie between 0 and +1 by an activation function to give the node's output (*Out*); this scaling is typically achieved using a logistic 'squashing' (or sigmoidal) function:

$$Out = \frac{1}{(1+\exp^{-Net})}$$

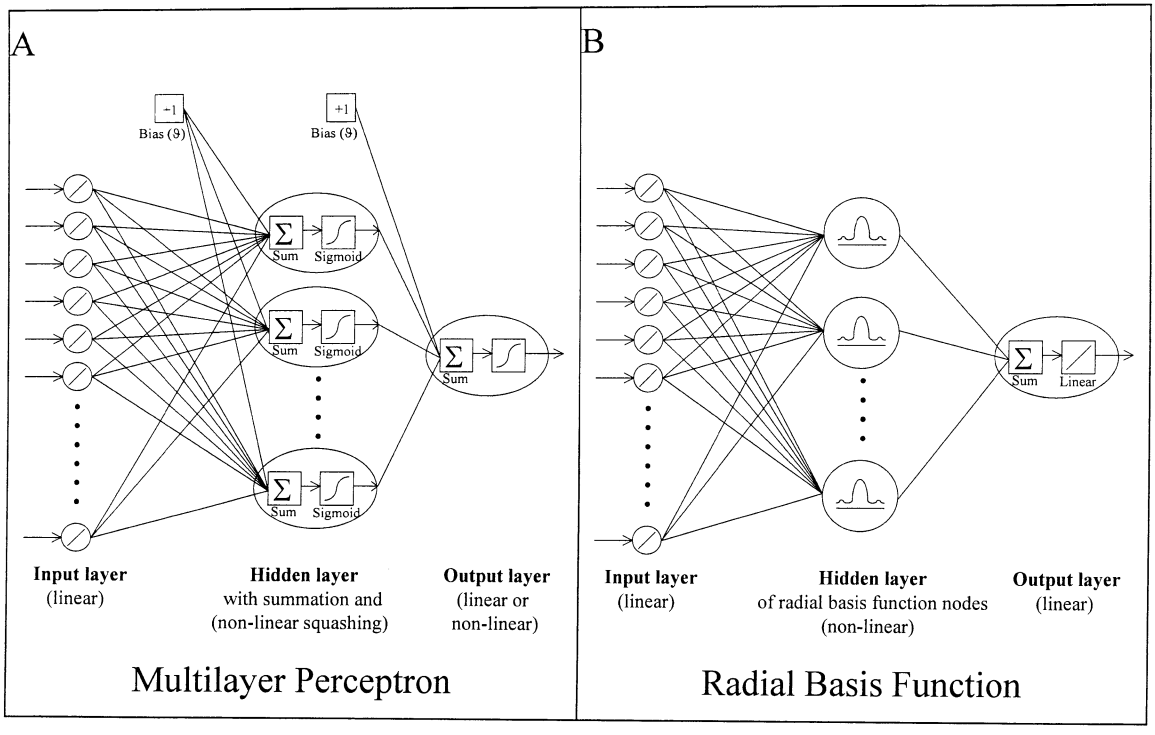

Fig. 6.10 (A) A multilayer perceptron neural network consisting of an input layer connected to a single node in the output layer by 1 hidden layer. In the architecture shown, adjacent layers of the network are fully interconnected although other architectures are possible. Nodes in the hidden and output layers consist of processing elements which sum the input applied to the node and scale the signal using a sigmoidal logistic squashing function. (B) Radial basis function neural network consisting of an input layer connected to a single node in the output layer by 1 hidden layer. The hidden layer consists of radially-symmetric Gaussian functions, although others exist (e.g., Mexican hat and thin plate splines).

These signals (*Out*) are then passed to the output node which sums them; in turn they are squashed by the logistic sigmoidal activation function; the product of this node is then fed to the 'outside world'.

For the training of the MLP the algorithm used most often is standard back-propagation (BP) (Chauvin and Rumelhart 1995; Haykin 1994; Rumelhart *et al.* 1986; Wasserman 1989; Werbos 1994). When input is applied to the network, it is allowed to run until an output is produced at each output node. The differences between the actual and the desired output, taken over the entire training set, are fed back through the network in the reverse direction to signal flow (hence back-propagation) modifying the weights as they go. This process is repeated until a suitable level of error is achieved.

One reason that MLPs are so attractive for the analysis of multivariate (spectral) data is that it has been shown mathematically (Cybenko 1989; Funabashi 1989; Hornik *et al.* 1989; Hornik *et al.* 1990; White 1990, 1992) that a MLP neural network consisting of only one hidden layer, with an arbitrarily large number of nodes, can learn any arbitrary (and hence non-linear) mapping of a continuous function to an arbitrary degree of accuracy.

6.2 Radial basis functions

By contrast RBFs networks are hybrid neural networks encompassing both unsupervised and supervised learning (Beale and Jackson 1990; Bishop 1995; Broomhead and Lowe 1988; Hush and Horne 1993; Moody and Darken 1989; Park and Sandberg 1991; Saha and Keller 1990; Walczak and Massart 1996; Wilkins *et al.* 1994*a*). RBFs are also typically three-layer neural networks and in essence the sigmoidal squashing function is replaced by non-linear (often either Gaussian or 'Mexican hat') basis functions or kernels (Figure 6.10(B)) The kernel is the function that determines the output of each node in the hidden layer when an input pattern is applied to it. This output is simply a function of the Euclidean distance from the kernel centre to the presented input pattern in the multi-dimensional space, and each node in the hidden layer only produces an output when the input applied is within its receptive field; if the input is beyond this receptive field the output is 0. This receptive field can be chosen and is radially symmetric around the kernel centre. Between them the receptive fields cover the entire region of the input space in which a multivariate input pattern may occur; a diagrammatic representation of this is given in Fig. 6.11, where a two-dimensional input is mapped by seven radially symmetric basis functions. This is a fundamentally different approach from the MLP, in which each hidden node represents a non-linear hyperplanar decision boundary bisecting the input space (Fig. 6.1). Thus RBF's have the advantage over gradient descent MLPs in that they have the ability to learn any arbitrary non-linear mapping of a *dis*-continuous function to an arbitrary degree of accuracy (Haykin 1994; Bishop 1995; Broomhead and Lowe 1988).

The outputs of the RBF nodes in the hidden layer are then fed forward via weighted connections to the nodes in the output layer in a similar fashion to the MLP, and each output node calculates a weighted sum of the outputs from the non-linear transfer from the kernels in the hidden layer. The only difference is that the output nodes of an RBF network are normally linear, whilst those of the MLP more typically employ a sigmoidal or logistic (non-linear) squashing function.

Thus in the RBF training proceeds in two stages: *Stage 1* involves unsupervised clustering of the input data, typically using the K-means clustering algorithm (Duda and Hart 1973; Everitt 1993; Hush and Horne 1993) to divide the high-dimensional input data into clusters. Next, kernel centres are placed at the mean of each cluster of data points. The use of K-means is particularly convenient because it positions the kernels relative to the density of the input data points. Next the receptive field is determined by the nearest neighbour heuristic where r_j (the radius of kernel j) is set to the Euclidean distance between w_j (the vector determining the centre for the j^{th} RBF) and its nearest neighbour (k), and an overlap constant (*Overlap*) is used:

$$r_j = Overlap \times \min (\|w_j - w_k\|)$$

where $\| \ldots \|$ denotes a vector norm, or Euclidean distance.

The overlap that often gives best results is where the edge of the radius of one kernel is at the centre of its nearest neighbour (Saha and Keller 1990).

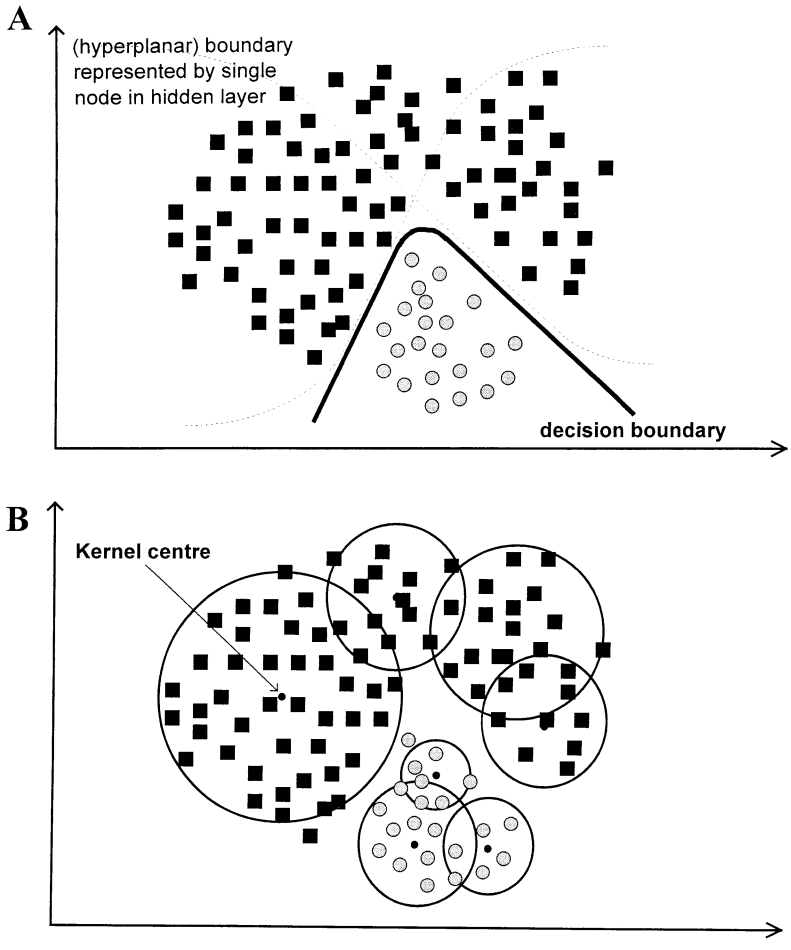

Fig. 6.11 (A) Typical decision boundary for a classification problem created between two data classes by a MLP with 2 nodes in the hidden layer, for 2 input nodes. Each hidden node represents a non-linear boundary and the nodes in the output layer interpolate this to form a decision boundary. (B) The same classification problem modelled by 7 radially symmetric basis functions. The width of each kernel function (referred to as its receptive field) is determined by the local density distribution of training examples.

The output from nodes in the hidden layer is dependent on the shape of the basis function and the one used was that of the Gaussian. Thus this value (R_j) for node j when given the i^{th} input vector (i_i) can be calculated by:

$$R_j(i_i) = \exp^{-\left(i_i^2 / r_j^2\right)}$$

Stage 2 involves supervised learning using simple linear regression. The inputs are the output values for all n basis functions ($R_1 - R_n$) for all the training input patterns to that layer ($i_1 - i_n$), and the outputs are identities binary encoded as shown in Fig. 6.9.

6.3 Identification of biological materials using spectroscopic measurements

Pyrolysis-MS involves the thermal degradation of non-volatile complex molecules in a vacuum causing their cleavage to smaller, volatile fragments separable by a mass spectrometer on the basis of their mass-to-charge ratio (*m/z*) (Goodacre and Kell 1996; Irwin 1982; Magee 1993; Meuzelaar *et al.* 1982). PyMS allows the chemically based discrimination of bacterial and fungal cells and produces complex biochemical fingerprints (i.e., pyrolysis mass spectra) which are distinct for different micro-organisms (Magee 1993; Goodacre 1994). The analytically useful multivariate data (see Fig. 6.12 for an example) are typically constituted by a set of 150 normalised intensities

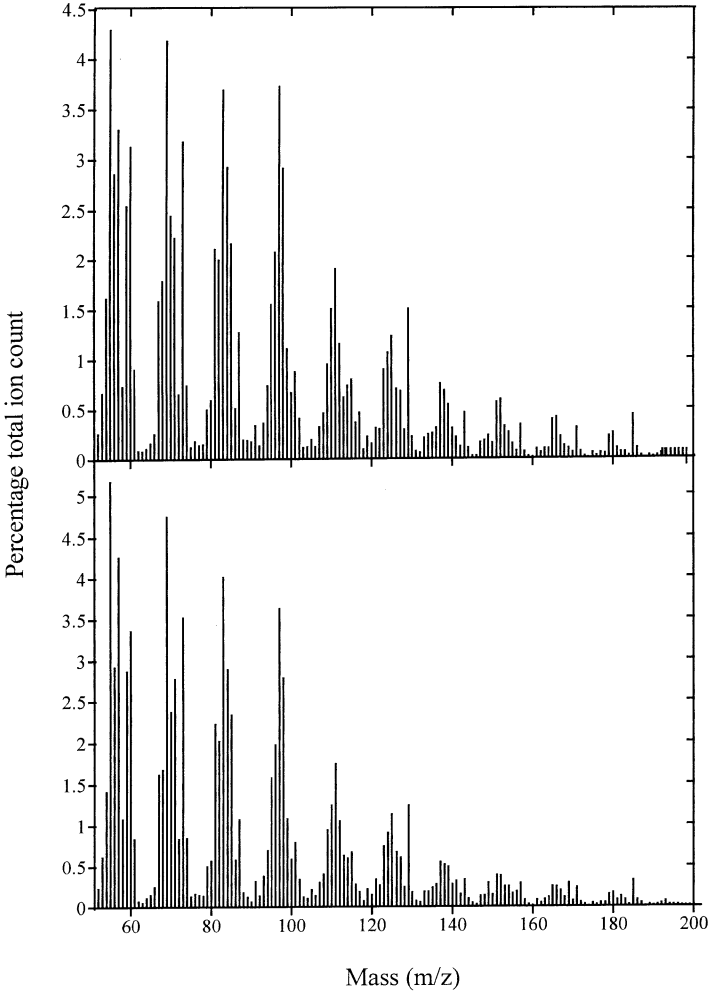

Fig. 6.12 Normalized pyrolysis mass spectra of *Propionibacterium acnes* from the foreheads of two healthy adult human individuals. *P. acnes* is a common inhabitant of the skin of humans, post puberty, and is considered to cause skin disorders and acne.

versus *m/z* in the range 51 to 200 and these are applied to the nodes on the input layers of ANNs.

The first demonstration of the ability of MLPs to discriminate between biological samples from their pyrolysis mass spectra was for the qualitative assessment of the adulteration of extra virgin olive oils with various seed oils (Goodacre *et al.* 1992, 1993*b*); in this study, which was performed double-blind, neural networks were trained with the spectra from 12 virgin olive oils, coded 1 at the output node, and with the spectra from 12 adulterated oils, which were coded 0. All oils in the test were correctly identified; in a typical run, the virgins were assessed with a code of 0.99976 ± 0.000146 (range 0.99954 – 1.00016) and the adulterated olive oils in the test set with a code of 0.001079 ± 0.002838 (range 0.00026 – 0.01009). This permitted their rapid and precise assessment, a task which previously was labour intensive and very difficult. It was most significant that the traditional 'unsupervised' multivariate analyses of principal component, discriminant, and cluster analyses failed to separate the oils according to whether they were pure or adulterated but rather discriminated them on the basis of the *type* of olive tree that the olive fruit came from.

The use of pyrolysis-MS with MLPs for the analysis of foodstuffs is becoming widespread and has been investigated for identifying the geographical origin of olive oils (Salter *et al.* 1997), for the characterization of cocoa butters (Anklam *et al.* 1997), and for differentiating between industrially made vinegar 'Aceto Balsamico di Modena' and traditionally produced vinegar 'Aceto Balsamico Tradizionale di Modena e di Reggio Emilia' (Anklam *et al.* 1998); the latter is often substituted with the industrial vinegar in an attempt to fool the consumer!

Several studies have also shown that this combination of PyMS and MLPs is also very effective for the rapid identification of a variety of bacterial strains of industrial, clinical, and veterinary importance. For example this approach has allowed the propionibacteria isolated from dogs to be correctly identified as human *Propionibacterium acnes* (Goodacre *et al.* 1994*b*), for detecting *Escherichia coli* isolates which produced verocytotoxins (Sisson *et al.* 1995), for distinguishing between *Mycobacterium tuberculosis* and *M. bovis* (Freeman *et al.* 1994), and for identifying streptomycetes recovered from soil (Chun *et al.* 1993*a, b*), oral abscess bacteria (Goodacre *et al.* 1996*e*), and fungi belonging to the genus *Penicillium* which were associated with cheese (Nilsson *et al.* 1996).

RBF neural networks have been rather less widely applied to the analysis of spectral data. Boddy and colleagues have used RBFs for the identification of marine phytoplankton from flow cytometric data (measurements based on: time of flight, light scattering, and fluorescence) (Morgan *et al.* 1998; Wilkins *et al.* 1994*b*; 1996). Other studies that have exploited RBFs include those based on PyMS to detect physiological changes in industrial fermentations of *Streptomyces* species (Kang *et al.* 1998*a,b*), for the classification of odours and food stuffs using chemical gas sensors in electronic nose arrays (Ping and Jun 1996; Schaller *et al.* 1998), for the correct discrimination of aromatic and non-aromatic chemical species from a 100 compound near-IR gas-phase library (Brown and Lo 1998), for the identification of common infectious

agents associated with urinary tract infection from their MS, IR, and Raman spectra (Goodacre *et al.* 1998*a*), and for the detection of cervical pre-cancer from fluorescence spectra from the cervix *in vivo* (Tumer *et al.* 1998).

With regard to neural network architecture other than the gradient descent and RBF-based algorithms illustrated above, Harrington (1993*a*) has compared minimal neural networks (MNN) with BP-ANNs for the analysis of tandem mass spectrometry data. MNN differ from BP-ANNs in that they use localized processing and build classification trees with branches composed of multiple processing units. A global entropy minimization may be achieved at a branch by combining the processing logic using principles from fuzzy set theory. Weight vectors are adjusted using an angular co-ordinate system and gradients of the fuzzy entropy function. The branches are optimal with respect to fuzziness and can accommodate non-linearly separable or ill-conditioned data. The most significant advantage of the MNNs is that relations among the training data and the mechanism of inference may be directly observed; thus rule-based classification trees have been constructed from the mass spectral daughter ions to discriminate between diesel smoke, dry yeast, *Escherichia coli*, MS-2 coliphage, grass pollen, *Bacillus subtilis*, fog oil, wood smoke, aldolase, and *Bacillus globigii* (Harrington 1993*b*).

7. Quantification

Quantification problems are those where the aim is to predict the amount of a substance. Spectral measurements of complex biological (organic) mixtures may be expressed, to some degree, in subpatterns of spectra describing the pure components of the mixtures and their relative concentrations; this is of course rather simplistic and spectral and biological interferences, from the total matrix, mean that the spectrum of A added to the spectrum of B does not equal the spectrum of (A + B). However, this complication aside, for present purposes, if a particular component A is changing in amount in a set of mixtures the subpattern spectrum of this component will also change magnitude relative to the amount of it; therefore multivariate calibration for the amount of A in the total spectra would be possible. This is a supervised learning problem, and the most commonly used machine learning method to achieve this is the MLP, where the output node(s) of these MLPs are encoded simply as the quantity of the component(s) to be measured.

Other non-cognitive supervised learning methods that are used for quantitative analyses include multiple linear regression (MLR), principal components regression (PCR), and partial least squares (PLS). PCR and PLS regression techniques are multivariate factor analysis methods (Jones *et al.* 1998; Liang *et al.* 1993; Martens and Næs 1989; Martin 1992) that are useful when the target matrix (here equivalent to the output layer of ANNs) does not contain the full model representation; that is to say, there are more variables in the data matrix than in the target matrix (which is generally the case with the spectroscopic measurements). As with supervised learning in MLPs both approaches utilize *a priori* information about the samples (Brereton 1990). The first stage in PCR is the decomposition

of the data (X-) matrix into latent variables by linear PCA; then each of the target (Y-) variables are regressed onto this decomposed X-matrix. PLS, however, performs a simultaneous and interdependent linear PCA decomposition in both X- and Y- matrices, in such a way that the information in the Y-matrix is used *directly* as a guide for the optimal decomposition of the X-matrix, and then performs linear regression of the latent variables on Y. It is considered that PLS usually handles several co-varying Y-variables better than does PCR, and is superior for the simultaneous modelling of several *intercorrelated* target variables (Martens and Næs 1989; Martin 1992).

The use of PLS and PCR for the deconvolution of spectroscopic data is well documented (Martens and Næs 1989). Indeed, studies comparing multiple least squares methods as well as the latent variable PCR and PLS methods (Carey *et al.* 1986; Geladi and Kowalski 1986; Joreskog and Wold 1982) have concluded that the best linear regression technique appears to be PLS. Whilst other studies based on a range of different spectroscopic data, *viz.*, fluorescence (McAvoy *et al.* 1992), infrared (Bhandare *et al.* 1994; Jacobsson and Hagman 1993), X-ray (Luo *et al.* 1997), mass spectra (Goodacre 1997; Goodacre *et al.* 1994*a*, 1995), measurements from a piezoelectric crystal sensor array (Xing and He 1997), and kinetic spectrophotometric determinations (Blanco *et al.* 1995, 1996) have concluded that ANNs often give better predictions than does PLS because ANNs are able to perform *non-linear* mappings of the inputs to output(s) whilst still being able to map the linear ones.

When trying to teach fully interconnected feedforward MLPs to quantify more than one thing at a time, it is best to have as many MLPs as components that are being quantified, since the main problem in teaching a single MLP with multiple outputs is that conflicting error messages are back-propagated from the output layer during the learning process (Bishop 1995; Jordan 1992). That is to say, the error that is fed back from one of the x output nodes is fed to *all* nodes in the preceding hidden layer, which also contains information pertinent to learning the other x–1 targets; if one target is failing to be learned, and thus sends the algorithm off in a different direction in weight space, it will inevitably hinder the learning of the other targets. This has been seen experimentally when comparing MLP predictions for the quantification of single and multicomponent mixtures from pyrolysis mass spectrometry (Goodacre *et al.* 1994*a*) and X-ray fluorescence measurements (Luo *et al.* 1998).

8. Interpretation of neural networks

The exploitation of novel multivariate analysis techniques employing ANNs which are based on *supervised* learning, rather than *unsupervised* methods, has permitted even better discrimination and quantification of biological systems. However, the information in terms of which input nodes are important is not *readily* available, and ANNs are often perceived as a 'black box' approach to modelling spectra. It is known from the statistical literature that

better predictions can often be obtained when only the most relevant input variables are considered (Bishop 1995; Miller 1990; Rawlings 1988; Ripley 1994; Ripley 1996; Seasholtz and Kowalski 1993). Therefore neural networks that prune larger networks are an active area of study (Finnoff *et al.* 1993; Hassibi and Stork 1993; LeCun *et al.* 1989; Mozer and Smolensky 1989; Reed 1993; Weigend *et al.* 1991), whilst it is also possible to grow neural networks from small ones (Broomhead and Lowe 1988; Fahlman and Lebiere 1990; Frean 1990; Moody and Darken 1989).

Alternatively PCA can be used to reduce the complexity of the MLP model. PCA is an excellent dimensionality reduction technique, since after the first few PCs are extracted subsequent ones will contribute only noise to the model. The use of PC scores as inputs to neural networks, without deterioration of the calibration model, has previously been applied to the analysis of UV/visible spectroscopic data (Blanco *et al.* 1995; Gemperline *et al.* 1991), for the identification of bacteria from their FT-IR spectra (Goodacre *et al.* 1998*b*; Goodacre *et al.* 1996*f*), for the quantification of bacteria (Timmins and Goodacre 1997) and foodstuffs (Goodacre 1997; Goodacre *et al.* 1997) from their PyMS spectra, and for the quantification of antibiotics in a bacterial matrix using FT-IR (Winson *et al.* 1997).

Another way to select the optimal number of inputs to a neural network is to use genetic algorithms (GAs) (Broadhurst *et al.* 1997). A GA is an optimization method based on the principles of Darwinian selection (Bäck *et al.* 1997; Goldberg 1989; Holland 1992; Mitchell 1995), and effectively performs a directed search through the multivariate space for possible solutions from random starting points. GAs are only one of a family of evolutionary computation methods, and perhaps a more powerful one is to use genetic programming (GP). GP is an application of the GA approach to derive mathematical equations, logical rules or program functions automatically (Gilbert *et al.* 1997; Koza 1992, 1994), and when applied to spectroscopic data have been shown to give very similar predictive results to ANNs but with the added benefit of spectral deconvolution in biochemical terms (Gilbert *et al.* 1997; Goodacre *et al.* 1998*c*; Taylor *et al.* 1998).

9. Concluding remarks

The application of neural networks for quantitative and qualitative analyses is well documented and accepted. ANNs clearly present themselves as extremely powerful and valuable tools for the analysis of multivariate data. Over the last few years the availability of powerful and inexpensive computers in conjunction with the development of user-friendly packages, which can simulate artificial neural networks, has led to machine learning increasingly being adopted by researchers in the biological, chemical, and physical sciences. Training a neural network is no longer cumbersome, and in the future it will be possible to devise automated cross validation techniques so that the network decides when it is optimally trained without user interference.

Acknowledgements

I am very grateful to Dr Richard Gilbert, Ms. Aoife McGovern, and Ms. Éadaoin Timmins for their constructive comments on this chapter, and to the Wellcome Trust for financial support (grant number 042615/Z/94/Z).

References

Anklam, E., Bassani, M.R., Eiberger, T., Kriebel, S., Lipp, M., and Matissek, R. (1997). Characterization of cocoa butters and other vegetable fats by pyrolysis mass spectrometry. *Fresenius Journal of Analytical Chemistry*,**357**, 981–4.

Anklam, E., Lipp, M., Radovic, B., Chiavaro, E., and Palla, G. (1998). Characterisation of Italian vinegar by pyrolysis mass spectrometry and a sensor device ('electronic nose'). *Food Chemistry,* **61**, 243–8.

Bäck, T., Fogel, D.B., and Michalewicz, Z. (1997). *Handbook of evolutionary computation*. IOP Publishing/Oxford University Press, Oxford.

Baxt, W.G. (1995). Application of artificial neural networks to clinical medicine. *The Lancet,* **346**, 1135–8.

Beale, R. and Jackson, T. (1990). *Neural computing: an introduction*. Adam Hilger, Bristol.

Berglund, A. and Wold, S. (1997). INLR, implicit non-linear latent variable regression. *Journal of Chemometrics,* **11**, 141–56.

Bhandare, P., Mendelson, Y., Stohr, E., and Peura, R.A. (1994). Glucose determination in simulated blood-serum solutions by Fourier-transform infrared spectroscopy — investigation of spectral interferences. *Vibrational Spectroscopy*, **6**, 363–78.

Bishop, C.M. (1995). *Neural networks for pattern recognition*. Clarendon Press, Oxford.

Blanco, M., Coello, J., Iturriaga, H., Maspoch, S., and Redon, M. (1995). Artificial neural networks for multicomponent kinetic determinations. *Analytical Chemistry*, **67**, 4477–83.

Blanco, M., Coello, J., Iturriaga, H., Maspoch, S., Redon, M., and Villegas, N. (1996). Artificial neural networks and partial least squares regression for pseudo-first-order with respect to the reagent multicomponent kinetic-spectrophotometric determinations. *Analyst,* **121**, 395–400.

Bowden, R., Mitchell, T.A., and Sarhadi, M. (1997). Cluster based nonlinear principle component analysis. *Electronics Letters*, **33**, 1858–9.

Brereton, R.G. (1990). *Chemometrics: applications of mathematics and statistics to laboratory systems*. Ellis Horwood, New York.

Brereton, R.G. (1992). *Multivariate pattern recognition in chemometrics.* Elsevier, Amsterdam.

Broadhurst, D., Goodacre, R., Jones, A., Rowland, J.J., and Kell, D.B. (1997). Genetic algorithms as a method for variable selection in PLS regression, with application to pyrolysis mass spectra. *Analytica Chimica Acta*, **348**, 71–86.

Broomhead, D.S. and Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems,* **2**, 312–55.

Brown, C.W. and Lo, S.C. (1998). Chemical information based on neural network processing of near-IR spectra. *Analytical Chemistry,* **70**, 2983–90.

Brown, S.D., Sum, S.T., Despagne, F., and Lavine, B.K. (1996). Chemometrics. *Analytical Chemistry*, **68**, R21–R61.

Carey, W.P., Beebe, K.R., Sanchez, E., Geladi, P., and Kowalski, B. (1986). Chemometric analysis of multisensor arrays. *Sensors and Actuators*, **9**, 223–34.

Causton, D.R. (1987). *A biologist's advanced mathematics*. Allen and Unwin, London.

Chatfield, C. and Collins, A.J. (1980). *Introduction to multivariate analysis*. Chapman and Hall, London.

Chauvin, Y. and Rumelhart, D.E. (1995). *Backpropagation: theory, architectures, and applications*. Erlbaum, Hove, UK.

Chun, J., Atalan, E., Ward, A.C., and Goodfellow, M. (1993*a*). Artificial neural network analysis of pyrolysis mass spectrometric data in the identification of *Streptomyces* strains. *FEMS Microbiology Letters*, **107**, 321–5.

Chun, J., Atalan, E., Kim, S.B., Kim, H.J., Hamid, M.E., Trujillo, M.E., Magee, J.G., Manfio, G.P., Ward, A.C., and Goodfellow, M. (1993*b*). Rapid identification of *Streptomycetes* by artificial neural network analysis of pyrolysis mass spectra. *FEMS Microbiology Letters*, **114**, 115–19.

Cybenko, G. (1989). Approximation by superposition of a sigmoidal function. *Mathematical Control Signals Systems*, **2**, 303–14.

Defernez, M. and Wilson, R.H. (1995). Midinfrared spectroscopy and chemometrics for determining the type of fruit used in jam. *Journal of the Science of Food and Agriculture* **67**, 461–7.

Duda, R.O. and Hart, P.E. (1973). *Pattern classification and scene analysis*. Wiley, New York.

Dybowski, R. and Gant, V. (1995). Artificial neural networks in pathological and medical laboratories. *Lancet*, **346**, 1203–7.

Erwin, E., Obermayer, K., and Schulten, K. (1992). Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetics*, **67**, 47–55.

Everitt, B.S. (1993). *Cluster analysis*. Edward Arnold, London.

Fahlman, S.E. and Lebiere, C. (1990). *The cascade-correlation learning architecture*. Report CMU-CS-90-100, Carnegie-Mellon University.

Finnoff, W., Hergert, F., and Zimmermann, H.G. (1993). Improving model selection by nonconvergent methods. *Neural Networks*, **6**, 771–83.

Frean, M. (1990). The upstart algorithm: a method for constructing and training feed-forward neural networks. *Neural Computation*, **2**, 198–209.

Freeman, R., Goodacre, R., Sisson, P.R., Magee, J.G., Ward, A.C., and Lightfoot, N.F. (1994). Rapid identification of species within the *Mycobacterium tuberculosis* complex by artificial neural network analysis of pyrolysis mass spectra. *Journal of Medical Microbiology*, **40**, 170–3.

Funahashi, K. (1989). On the approximate realization of continuous-mappings by neural networks. *Neural Networks*, **2**, 183–92.

Geladi, P. and Kowalski, B. (1986). Partial least-squares regression — a tutorial. *Analytica Chimica Acta*, **185**, 1–17.

Gemperline, P.J., Long, J.R., and Gregoriou, V.G. (1991). Nonlinear multivariate calibration using principal components regression and artificial neural networks. *Analytical Chemistry*, **63**, 2313–23.

Gilbert, R.J., Goodacre, R., Woodward, A.M., and Kell, D.B. (1997). Genetic programming: a novel method for the quantitative analysis of pyrolysis mass spectral data. *Analytical Chemistry*, **69**, 4381–9.

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, Reading, Mass.

Goodacre, R. (1994). Characterisation and quantification of microbial systems using pyrolysis mass spectrometry: Introducing neural networks to analytical pyrolysis. *Microbiology Europe*, **2**, 16–22.

Goodacre, R. (1997). Use of pyrolysis mass spectrometry with supervised learning for the assessment of the adulteration of milk of different species. *Applied Spectroscopy*, **51**, 1144–53.

Goodacre, R. and Kell, D.B. (1993). Rapid and quantitative analysis of bioprocesses using pyrolysis mass spectrometry and neural networks — application to indole production. *Analytica Chimica Acta*, **279**, 17–26.

Goodacre, R. and Kell, D.B. (1996). Pyrolysis mass spectrometry and its applications in biotechnology. *Current Opinion in Biotechnology*, **7**, 20–8.

Goodacre, R., Kell, D.B., and Bianchi, G. (1992). Neural networks and olive oil. *Nature*, **359**, 594.

Goodacre, R., Edmonds, A.N., and Kell, D.B. (1993*a*). Quantitative analysis of the pyrolysis mass spectra of complex mixtures using artificial neural networks — application to amino acids in glycogen. *Journal of Analytical and Applied Pyrolysis*, **26**, 93–114.

Goodacre, R., Kell, D.B., and Bianchi, G. (1993*b*). Rapid assessment of the adulteration of virgin olive oils by other seed oils using pyrolysis mass spectrometry and artificial neural networks. *Journal of the Science of Food and Agriculture*, **63**, 297–307.

Goodacre, R., Neal, M.J., and Kell, D.B. (1994*a*). Rapid and quantitative analysis of the pyrolysis mass spectra of complex binary and tertiary mixtures using multivariate calibration and artificial neural networks. *Analytical Chemistry*, **66**, 1070–85.

Goodacre, R., Neal, M.J., Kell, D.B., Greenham, L.W., Noble, W.C., and Harvey, R.G. (1994*b*). Rapid identification using pyrolysis mass spectrometry and artificial neural networks of *Propionibacterium acnes* isolated from dogs. *Journal of Applied Bacteriology*, **76**, 124–34.

Goodacre, R., Trew, S., Wrigley-Jones, C., Saunders, G., Neal, M.J., Porter, N., and Kell, D.B. (1995). Rapid and quantitative analysis of metabolites in fermentor broths using pyrolysis mass spectrometry with supervised learning: application to the screening of *Penicillium chrysogenum* fermentations for the overproduction of penicillins. *Analytica Chimica Acta*, **313**, 25–43.

Goodacre, R., Howell, S.A., Noble W.C., and Neal, M.J. (1996*a*). Sub-species discrimination using pyrolysis mass spectrometry and self-organising neural networks of *Propionibacterium acnes* isolated from normal human skin. *Zentralblatt für Bakteriologie — International Journal of Medical Microbiology, Virology, Parasitology and Infectious Diseases*, **284**, 501–15.

Goodacre, R., Pygall, J., and Kell, D.B. (1996*b*). Plant seed classification using pyrolysis mass spectrometry with unsupervised learning; the application of auto-associative and Kohonen artificial neural networks. *Chemometrics and Intelligent Laboratory Systems*, **34**, 69–83.

Goodacre, R., Rischert, D.J., Evans, P.M., and Kell, D.B. (1996*c*). Rapid authentication of animal cell lines using pyrolysis mass spectrometry and auto-associative artificial neural networks. *Cytotechnology*, **21**, 231–41.

Goodacre, R., Neal, M.J., and Kell, D.B. (1996*d*). Quantitative analysis of multivariate data using artificial neural networks: a tutorial review and applications to the deconvolution of pyrolysis mass spectra. *Zentralblatt für Bakteriologie —*

International Journal of Medical Microbiology, Virology, Parasitology and Infectious Diseases, **284,** 516–39.

Goodacre, R., Hiom, S.J., Cheeseman, S.L., Murdoch, D., Weightman, A.J., and Wade, W.G. (1996*e*). Identification and discrimination of oral asaccharolytic *Eubacterium* spp. using pyrolysis mass spectrometry and artificial neural networks. *Current Microbiology,* **32**, 77–84.

Goodacre, R., Timmins, É.M., Rooney, P.J., Rowland, J.J., and Kell, D.B. (1996*f*). Rapid identification of *Streptococcus* and *Enterococcus* species using diffuse reflectance-absorbance Fourier transform infrared spectroscopy and artificial neural networks. *FEMS Microbiology Letters,* **140**, 233–9.

Goodacre, R., Hammond, D., and Kell, D.B. (1997). Quantitative analysis of the adulteration of orange juice with sucrose using pyrolysis mass spectrometry and chemometrics. *Journal of Analytical and Applied Pyrolysis,* **40/41**, 135–58.

Goodacre, R., Timmins, E.M., Burton, R., Kaderbhai, N., Woodward, A.M., Kell, D.B., and Rooney, P.J. (1998*a*). Rapid identification of urinary tract infection bacteria using hyperspectral, whole organism fingerprinting and artificial neural networks. *Microbiology,* **144**, 1157–70.

Goodacre, R., Rooney, P.J., and Kell, D.B. (1998*b*). Discrimination between methicillin-resistant and methicillin-susceptible *Staphylococcus aureus* using pyrolysis mass spectrometry and artificial neural networks. *Journal of Antimicrobial Chemotherapy,* **41**, 27–34.

Goodacre, R., Shann, B., Gilbert, R.J., Timmins, É.M., McGovern, A.C., Alsberg, B.K., Logan, N.A., and Kell, D.B. (1998*c*). The characterisation of *Bacillus* species from PyMS and FT IR data. In *Proc. 1997 ERDEC scientific conference on chemical and biological defense research*, ERDEC-SP-063, pp. 257–265.

Gutteridge, C.S., Vallis, L., and MacFie, H.J.H. (1985). Numerical methods in the classification of microorganisms by pyrolysis mass spectrometry. In *Computer-assisted bacterial systematics,* pp. 369–401. Edited by M. Goodfellow, D. Jones, and F. Priest. Academic Press, London.

Harrington, P.D. (1993*a*). Minimal neural networks — concerted optimization of multiple decision planes. *Chemometrics and Intelligent Laboratory Systems,* **18**, 157–70.

Harrington, P.B. (1993*b*). Minimal neural networks: differentiation of classification entropy. *Chemometrics and Intelligent Laboratory Systems,* **19**, 143–54.

Hassibi, B. and Stork, D.G. (1993). Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems 5*, pp. 164–71. Edited by S.J. Hanson, J.D. Cowan, and J.D. Giles. Morgan Kaufmann, San Mateo, CA.

Haykin, S.S. (1994). *Neural networks: a comprehensive foundation.* Macmillan, New York.

Hecht-Nielsen, R. (1990). *Neurocomputing*. Addison-Wesley, Massachusetts.

Heikka, R.A., Immonen, K.T., Minkkinen, P.O., Paatero, E.Y.O., and Salmi, T.O. (1997). Determination of acid value, hydroxyl value and water content in reactions between dicarboxylic acids and diols using near-infrared spectroscopy and non-linear partial least squares regression. *Analytica Chimica Acta,* **349**, 287–94.

Hertz, J., Krogh, A., and Palmer, R.G. (1991). *Introduction to the theory of neural computation*. Addison-Wesley, California.

Holland, J.H. (1992). *Adaption in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT Press.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks,* **2**, 359–68.

Hornik, K., Stinchcombe, M., and White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks,* **3**, 551–60.

Höskuldsson, A. (1992). Quadratic PLS regression. *Journal of Chemometrics,* **6**, 307–34.

Hush, D.R. and Horne, B.G. (1993). Progress in supervised neural networks — what's new since Lippmann. *IEEE Signal Processing Magazine,* **10**, 8–39.

Irwin, W.J. (1982). *Analytical pyrolysis: a comprehensive guide.* Marcel Dekker, New York.

Jacobsson, S.P. (1994). Feature extraction of polysaccharides by low-dimensional internal representation neural networks and infrared spectroscopy. *Analytica Chimica Acta,* **291**, 19–27.

Jacobsson, S.P. and Hagman, A. (1993). Chemical-composition analysis of carrageenans by infrared spectroscopy using partial least squares and neural networks. *Analytica Chimica Acta,* **284**, 135–47.

Jolliffe, I.T. (1986). *Principal component analysis.* Springer-Verlag, New York.

Jones, A., Shaw, A.D., Salter, G.J., Bianchi, G., and Kell, D.B. (1998). The exploitation of chemometric methods in the analysis of spectroscopic data: application to olive oils. In *Lipid analysis of oils and fats,* pp. 317–76. Edited by R.J. Hamilton. Chapman and Hall, London.

Jordan, M.I. (1992). Constrained supervised learning. *Journal of Mathematical Psychology,* **36**, 396–425.

Joreskog, K. and Wold, H. (1982). *Systems under direct observation.* North Holland, Amsterdam.

Kaartinen, J., Hiltunen, Y., Kovanen, P.T., and AlaKorpela, M. (1998). Application of self-organizing maps for the detection and classification of human blood plasma lipoprotein lipid profiles on the basis of H^1 NMR spectroscopy data. *NMR in Medicine,* **11**, 168–76.

Kang, S.G., Kenyon, R.G.W., Ward, A.C., and Lee, K.J. (1998*a*). Analysis of differentiation state in *Streptomyces albidoflavus* SMF301 by the combination of pyrolysis mass spectrometry and neural networks. *Journal of Biotechnology,* **62**, 1–10.

Kang, S.G., Lee, D.H., Ward, A.C., and Lee, K.J. (1998*b*). Rapid and quantitative analysis of clavulanic acid production by the combination of pyrolysis mass spectrometry and artificial neural network. *Journal of Microbiology and Biotechnology,* **8**, 523–30.

Kell, D.B. and Sonnleitner, B. (1995). GMP — Good Modelling Practice: an essential component of good manufacturing practice. *Trends in Biotechnology,* **13**, 481–92.

Kohonen, T. (1989). *Self-organization and associative memory.* Springer-Verlag, Berlin.

Koza, J.R. (1992). *Genetic programming: on the programming of computers by means of natural selection.* MIT Press, Cambridge, MA.

Koza, J.R. (1994). *Genetic programming II: automatic discovery of reusable programs.* MIT Press, Cambridge, MA.

Kramer, M.A. (1991). Non linear principal components analysis using auto-associative neural networks. *AIChE Journal,* **37**, 233–43.

Kramer, M.A. (1992). Autoassociative neural networks. *Computers & Chemical Engineering,* **16**, 313–28.

Kuespert, D.R. and McAvoy, T.J. (1994). Knowledge extraction in chemical process control. *Chemical Engineering Communications,* **130**, 251–64.

Kvalheim, O.M., Aksnes, D.W., Brekke, T., Eide, M.O., Sletten, E., and Telnæs, N. (1985). Crude oil characterization and correlation by principal component analysis of ^{13}C nuclear magnetic resonance spectra. *Analytical Chemistry,* **57**, 2858–64.

Lavine, B.K. (1998). Chemometrics. *Analytical Chemistry,* **70**, R209–R228.

LeCun, Y., Denker, J.S., and Solla, S.A. (1989). Optimal brain damage. In *Advances in neural information processing systems 1*, pp. 598–605. Edited by D.S. Touretzky. Morgan Kaufmann, San Mateo, CA.

Leonard, J.A and Kramer, M.A. (1993). Diagnosing dynamic faults using modular neural nets. *IEEE Expert Intelligent Systems & their Applications,* **8**, 44–53.

Liang, Y.Z. Kvalheim, O.M., and Manne, R. (1993). White, grey and black multicomponent systems — a classification of mixture problems and methods for their quantitative-analysis. *Chemometrics and Intelligent Laboratory Systems*, **18**, 235–50.

Luo, L.Q., Guo, C.L., Ma, GZ., and Ji, A. (1997). Choice of optimum model parameters in artificial neural networks and applications to x-ray fluorescence analysis. *X-Ray Spectrometry*, **26**, 15–21.

Luo, L.Q., Ji, A., Ma, G.Z., and Guo, C.L. (1998). Focusing on one component each time — comparison of single and multiple component prediction algorithms in artificial neural networks for x-ray fluorescence analysis. *X-Ray Spectrometry* **27**, 17–22.

MacFie, H.J.H., Gutteridge, C.S., and Norris, J.R. (1978). Use of canonical variates in differentiation of bacteria by pyrolysis gas-liquid chromatography. *Journal of General Microbiology,* **104**, 67–74.

Magee, J.T. (1993). Whole-organism fingerprinting. In *Handbook of new bacterial systematics*, pp. 383–427. Edited by M. Goodfellow and A.G. O'Donnell. Academic Press, London.

Manly, B.F.J. (1994). *Multivariate statistical methods: a primer.* Chapman and Hall, London.

Mark, H. (1991). *Principles and practice of spectroscopic calibration.* John Wiley & Sons, New York.

Martens, H. and Næs, T. (1989). *Multivariate calibration.* John Wiley, Chichester.

Martin, K.A. (1992). Recent advances in near-infrared reflectance spectroscopy. *Applied Spectroscopy Reviews,* **27**, 325–83.

Massart, D.L., Vandeginste, B.G.M., Deming, S.N., Michotte, Y., and Kaufmann, L. (1988). *Chemometrics: a textbook.* Elsevier, Amsterdam.

McAvoy, T.J., Su, H.T., Wang, N.S., He, M., Horvath, J., and Semerjian, H. (1992). A comparison of neural networks and partial least-squares for deconvoluting fluorescence spectra. *Biotechnology and Bioengineering,* **40**, 53–62.

Meuzelaar, H.L.C., Haverkamp, J., and Hileman, F.D. (1982). *Pyrolysis mass spectrometry of recent and fossil biomaterials.* Elsevier, Amsterdam.

Miller, A.J (1990). *Subset selection in regression.* Chapman and Hall, London.

Minsky, M.L. and Papert, S.A. (1969). *Perceptrons.* MIT Press, Cambridge, MA.

Mitchell, M. (1995). *An introduction to genetic algorithms.* MIT Press, Boston.

Montague, G. and Morris, J. (1994). Neural network contributions in biotechnology. *Trends In Biotechnology,* **12**, 312–24.

Moody, J. and Darken, C.J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation,* **1**, 281–94.

Morgan, A., Boddy, L., Mordue, J.E.M., and Morris, C.W. (1998). Evaluation of artificial neural networks for fungal identification, employing morphometric data from spores of Pestalotiopsis species. *Mycological Research,* **102**, 975–84.

Mozer, M.C. and Smolensky, P. (1989). Skeletonization: a technique for trimming fat from a network *via* relevance assessment. In *Advances in neural information processing systems 1,* pp. 107–15. Edited by D.S. Touretzky. Morgan Kaufmann, San Mateo, CA.

Musil, M. and Plesinger, A. (1996). Discrimination between local microearthquakes and quarry blasts by multi-layer perceptrons and Kohonen maps. *Bulletin of the Seismological Society of America,* **86**, 1077–90.

Nilsson, T., Bassani, M.R., Larsen, T.O., and Montanarella, L. (1996). Classification of species in the genus *Penicillium* by Curie point pyrolysis mass spectrometry followed by multivariate analysis and artificial neural networks. *Journal of Mass Spectrometry,* **31**, 1422–8.

Park, J. and Sandberg, I.W. (1991). Universal approximation using radial basis function networks. *Neural Computation,* **3**, 246–57.

Parker, D.B. (1982). Learning-logic. Invention report, S81-64, File 1. Office of Technology Licensing, Stanford University.

Ping, W. and Jun, X. (1996). A novel recognition method for electronic nose using artificial neural network and fuzzy recognition. *Sensors and Actuators B-Chemical,* **37**, 169–74.

Rawlings, J.O. (1988). *Applied regression analysis.* Wadsworth & Brooks, Pacific Grove, CA.

Reed, R. (1993). Pruning algorithms — a survey. *IEEE Transactions in Neural Networks,* **4**, 740–7.

Richard, M.D. and Lippmann, R.P. (1991). Neural network classifiers estimate Bayesian *a posteriori* probabilities. *Neural Computation,* **3**, 461–83.

Ripley, B.D. (1994). Neural networks and related methods for classification. *Journal of the Royal Statistical Society, Series B-Methodological,* **56**, 409–37.

Ripley, B.D. (1996). *Pattern recognition and neural networks.* Cambridge University Press, Cambridge.

Rumelhart, D.E., McClelland, J.L., and The PDP Research Group (1986). *Parallel distributed processing, experiments in the microstructure of cognition.* MIT Press, Cambridge, Mass.

Saha, A. and Keller, J.D. (1990). Algorithms for better representation and faster learning in radial basis functions. In *Advances in neural information processing systems,* pp. 482–9 Edited by D. Touretzky. Morgan Kaufmann Publishers, San Mateo, CA.

Salter, G.J., Lazzari, M., Giansante, L., Goodacre, R., Jones, A., Surricchio, G., Kell, D.B., and Bianchi, G. (1997). Determination of the geographical origin of Italian extra virgin olive oil using pyrolysis mass spectrometry and artificial neural networks. *Journal of Analytical and Applied Pyrolysis,* **40/41**, 159–70.

Schaller, E., Bosset, J.O. and Escher, F. (1998). 'Electronic noses' and their application to food. *Food Science and Technology-Lebensmittel-Wissenschaft & Technologie,* **31**, 305–16.

Seasholtz, M.B. and Kowalski, B. (1993). The parsimony principle applied to multivariate calibration. *Analytica Chimica Acta,* **277**, 165–77.

Simpson, P.K. (1990). *Artificial neural systems.* Pergamon Press, Oxford.

Sisson, P.R., Freeman, R., Law, D., Ward, A.C., and Lightfoot, N.F. (1995). Rapid detection of verocytotoxin production status in *Escherichia coli* by artificial neural

network analysis of pyrolysis mass spectra. *Journal of Analytical and Applied Pyrolysis,* **32**, 179–85.

Stamkopoulos, T., Diamantaras, K., Maglaveras, N., and Strintzis, M. (1998). ECG analysis using nonlinear PCA neural networks for ischemia detection. *IEEE Transactions of Signal Processing,* **46**, 3058–67.

Taavitsainen, V.M. and Korhonen, P. (1992). Nonlinear data analysis with latent variable. *Chemometrics and Intelligent Laboratory Systems,* **14**, 185–94.

Taylor, J., Goodacre, R., Wade, W.G., Rowland, J.J., and Kell, D.B. (1998). The deconvolution of pyrolysis mass spectra using genetic programming: application to the identification of some *Eubacterium* species. *FEMS Microbiology Letters,* **160**, 237–46.

Timmins, É.M. and Goodacre, R. (1997). Rapid quantitative analysis of binary mixtures of *Escherichia coli* strains usign pyrolysis mass spectrometry with multivariate calibration and artificial neural networks. *Journal of Applied Microbiology,* **83**, 208–18.

Tumer, K., Ramanujam, N., Ghosh, J., and Richards-Kortum, R. (1998). Ensembles of radial basis function networks for spectroscopic detection of cervical precancer. *IEEE Transactions on Biomedical Engineering,* **45**, 953–61.

Tzovaras, D. and Strintzis, M.G. (1998). Use of nonlinear principal component analysis and vector quantization for image coding. *IEEE transactions on image processing,* **7**, 1218–23.

Walczak, B. and Massart, D.L. (1996). The radial basis functions — partial least squares approach as a flexible non-linear regression technique. *Analytical Chimica Acta,* **331**, 177–85.

Waldemark, J. (1997). An automated procedure for cluster analysis of multivariate satellite data. *International Journal of Neural Systems,* **8**, 3–15.

Wasserman, P.D. (1989). *Neural computing: theory and practice.* Van Nostrand Reinhold, New York.

Weigend, A.S., Rumelhart, D.E., and Huberman, B.A. (1991). Generalization by weight-elimination with application to forecasting. In *Neural information processing systems,* pp. 875–82. Edited by R.P. Lippmann, E. Moody, and D.S. Touretzky. Morgan Kaufmann, San Mateo, CA.

Werbos, P.J. (1994). *The roots of back-propagation: from ordered derivatives to neural networks and political forecasting.* John Wiley, Chichester.

White, H. (1990). Connectionist nonparametric regression — multilayer feedforward networks can learn arbitrary mappings. *Neural Networks,* **3**, 535–49.

White, H. (1992). *Artificial neural networks: approximation and learning theory.* Blackwell, Oxford.

Wilkins, M.F., Boddy, L., and Morris, C.W. (1994*a*). Kohonen maps and learning vector quantization neural networks for analysis of multivariate biological data. *Binary Computing in Microbiology,* **6**, 64–72.

Wilkins, M.F., Morris, C.W., and Boddy, L. (1994*b*). A comparison of radial basis function and backpropagation neural networks for identification of marine phytoplankton from multivariate flow cytometry data. *Computer Applications In the Biosciences* **10**, 285–94.

Wilkins, M.F., Boddy, L., Morris, C.W., and Jonker, R. (1996). A comparison of some neural and non-neural methods for identification of phytoplankton from flow cytometry data. *Computer Applications In the Biosciences,* **12**, 9–18.

Windig, W., Haverkamp, J., and Kistemaker, P.G. (1983). Interpretation of sets of pyrolysis mass spectra by discriminant analysis and graphical rotation. *Analytical Chemistry,* **55**, 81–8.

Winson, M.K., Goodacre, R., Woodward, A.M., Timmins, É.M., Jones, A., Alsberg, B.K., Rowland, J.J., and Kell, D.B. (1997). Diffuse reflectance absorbance spectroscopy taking in chemometrics (DRASTIC). A hyperspectral FT-IR-based approach to rapid screening for metabolite overproduction. *Analytica Chimica Acta*, **348**, 273–82.

Wold, S., Kettaneh-Wold, N., and Skagerberg, B. (1989). Nonlinear PLS modelling. *Chemometrics and Intelligent Laboratory Systems*, **7**, 53–65.

Wold, S. (1992). Nonlinear partial least-squares modeling .2. spline inner relation. *Chemometrics and Intelligent Laboratory Systems*, **14**, 71–84.

Xing, W.L. and He, X.W. (1997). Crown ether-coated piezoelectric crystal sensor array for detection of organic vapour mixtures using several chemometric methods. *Analyst*, **122**, 587–91.

Zupan, J. and Gasteiger, J. (1993). *Neural networks for chemists: an introduction.* VCH Verlagsgeesellschaft, Weinheim.

7 Applications of knowledge-based systems

Mary Mulholland and D. Brynn Hibbert

1. Introduction

Knowledge-based systems are at the heart of many Artificial Intelligence (AI) approaches. To answer high-level questions in science requires human expertise that resides in the experience of a practitioner. The sum of that experience comprises *knowledge*, and applications that rely on this amassed knowledge are referred to as knowledge-based systems. Expert systems (ESs) are particular knowledge-based applications that use some form of inference engine to advise a user via a suitable interface (Fig. 7.1).

Research concentrates on the nature of the inference engine, the construction and maintenance of the knowledge base in a form that is useable by the inference engine, and the user interface. There have been a great number of expert systems published, too many to adequately review in one chapter. In chemistry alone 1500 publications have appeared since 1984. Therefore, we shall focus on the area of analytical chemistry. First, we discuss practical aspects of the construction of knowledge-based expert systems and review a number of systems in analytical chemistry. A novel approach to knowledge engineering and maintenance of an ES, using Ripple Down Rules (RDR), and its application to ion chromatography is then described. Finally we shall discuss reasons why, with the obvious research interest in ES, there are so few working systems in use.

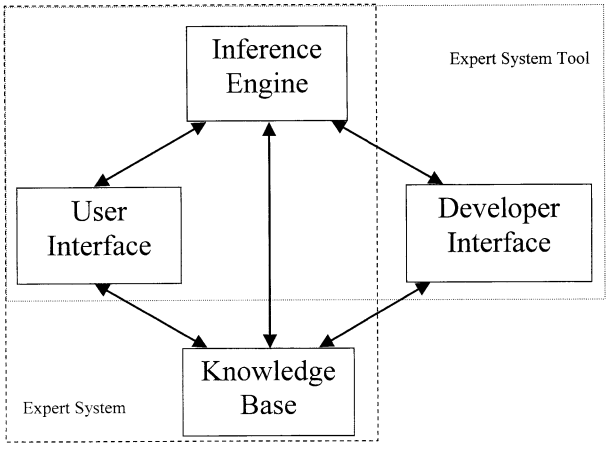

Fig. 7.1 The components of an Expert System and an Expert System development tool.

2. Ion chromatography

Throughout this chapter, the analytical method of liquid chromatography (also known as high pressure liquid chromatography or HPLC) and its sub-method, ion chromatography (IC), will be used as an example of a widely used method to which expert system technologies may be applied. IC is an analytical method that separates and detects ions in solution as a means of identifying and quantifying them (Haddad and Jackson 1990). A small volume of sample is injected into a flowing stream (the eluent) which passes through a column in which the separation is effected. The ions issuing from the column pass into a detector, the signal from which is displayed as a series of peaks. The time from injection to the appearance of a peak is characteristic of a given ion, and the peak height or area is proportional to its concentration in the sample. Chemical analysis using IC requires a variety of expertise which would not be possible to encompass in one ES. In breaking down the domain into smaller, more manageable parts, it is first necessary to decide how to represent the various research areas of IC.

Figure 7.2 shows the stages in the development of a chromatographic method. The first stage is to collect information on the number and type of ions, the complexity of the sample matrix, and the requirements of the application. A check is usually made of the literature to find a previously developed method, and if such a method is available, it can be configured to a complete description. Alternatively, the method may require some further optimization. If a method is not available then a first guess is made on a suitable sample preparation technique, column, eluent, pH, and detector. This can be further optimized for retention, selectivity, and instrumental conditions. The calibration technique can be chosen and the final method validated. Selection of the calibration method is often an integral part of the validation study.

Each stage shown in Fig. 7.2 may be selected for an ES application. Maris and Hindriks (1993) present a similar scheme, and suggest that four aspects of the development of a chromatographic method are amenable to an ES: the initial guess of the configuration, optimization of selectivity, optimization of chromatographic conditions, and validation.

The initial configuration presents a classical problem of disparate and often incomplete knowledge about the system with an expert's general knowledge, aided by the literature, to give a workable system within the constraints of the laboratory. Traditional expert systems have been written to advise on the configuration and the novel method of ripple down rules is described below.

Optimization of selectivity and chromatographic conditions is a problem treated by AI, but not necessarily knowledge-based systems. Several algorithmic optimization methods have been used to find, for example, the best composition of the mobile phase. Here propose and revise strategies work well as new combinations may be tried experimentally.

Finally, validation is a process in which the method is proved to be 'fit for purpose' in terms of a number of analytical and statistical measures. The individual steps in validation are well known, but a knowledge-based system can

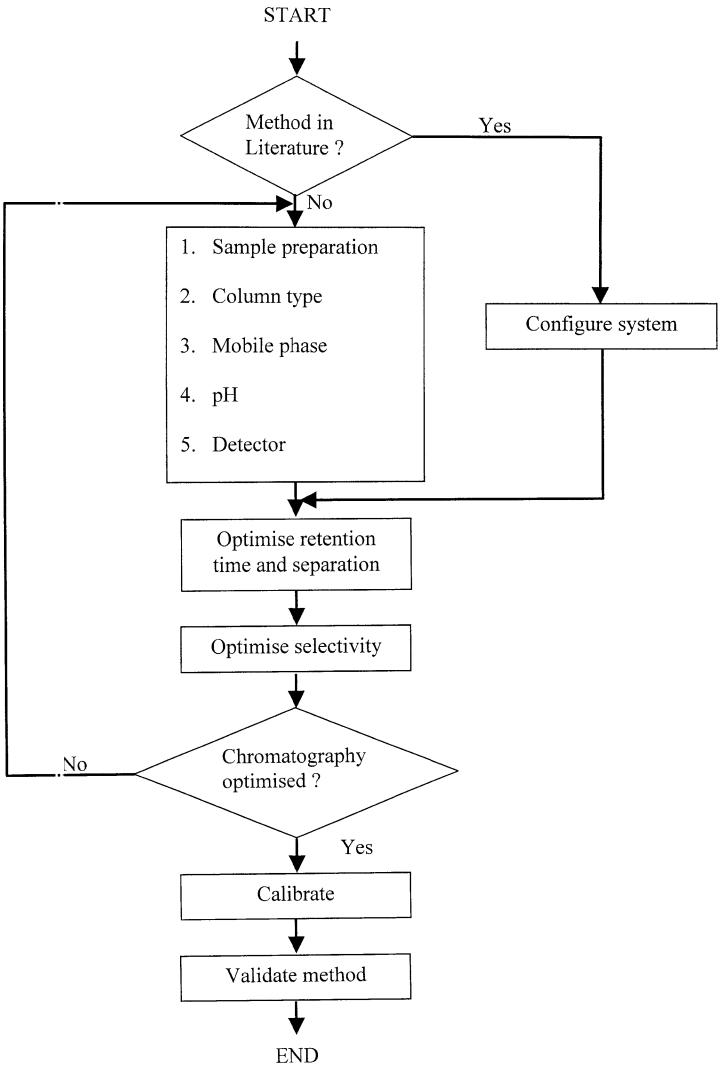

Fig. 7.2 Stages in the development of a chromatographic method.

lead the analyst through the necessary experiments, acting as an intelligent agent.

A major effort by the European Union to assemble software to accomplish some or all of these tasks will be described later.

The analytical chemistry method of liquid chromatography is sufficiently broad that it is not feasible to contemplate a single ES for all of the above tasks and all possible scenarios. However, particular needs can be accommodated by sacrificing breadth of application or depth of knowledge. A broad but shallow system tackles a large variety of problems but restrictions in computer hardware and software necessarily mean these systems do not contain detailed knowledge. The problem with broad systems is that users consult them only for their own application areas and very quickly learn the

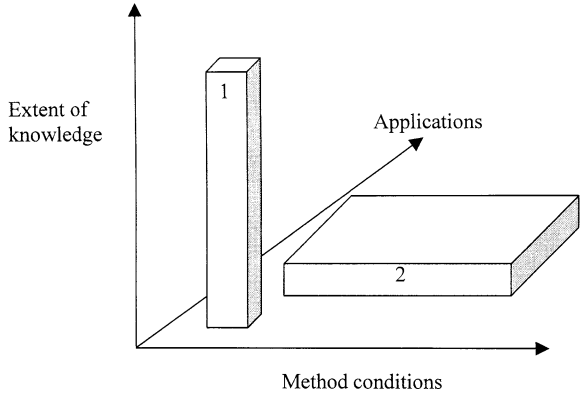

Fig. 7.3 Examples of 1: 'narrow and deep' and 2: 'broad and shallow' knowledge in the domain of ion chromatography.

contents of the knowledge base, rendering them useful only for a short time. The alternative is to build a narrow yet deep system that can tackle a small number of problems but in greater depth. However, their restricted ambit limits these systems to a small number of users. Taking the example of ion chromatography, the IC domain may be represented in two dimensions: *applications*: e.g., water analysis, pharmaceutical excipients, industrial effluent; and *method conditions*: e.g., the type of detector, column/eluent combinations, the separation mechanism. Figure 7.3 shows two possible domain definitions along three dimensions comprising the two IC dimensions described above and the extent of knowledge required. The first illustrates a narrow and deep domain, which would be the case of an ES developed for a pharmaceutical analyst who occasionally requires ion chromatography to assay ionic excipients. The ES would be required to cover a limited range of applications and separation mechanisms (only those suited to the available equipment) thus would be narrow. However, the knowledge of these separation mechanisms would have to be deep, as the analyst needs detailed information of the method conditions. This ES would have lasting use for a limited number of users, since it would take longer for the user to become familiar with the knowledge in the ES that is relevant to his/her domain.

The second example, a broad and shallow system, could be an ES for an analyst who is required to specialize in ion chromatography. The ES must contain knowledge of a wide range of applications and possible method conditions. Hence, by necessity, it must be shallow in detailed knowledge. The user needs to supplement the details, or in-depth knowledge, from the literature and eventually from their own experience. This system has a large number of potential users, but they would quickly learn the knowledge in the ES that is relevant to their domain and so the lifetime of this system is likely to be short.

The knowledge that is required for a system of any depth needs to be sufficiently complete and predictable to give acceptable accuracy and consistency of the method prediction. For a shallow system, this is not critical as

Table 7.1 A breakdown of the stages of building an ES (Goulder *et al.* 1988)

Conventional tasks	Time scale (months) based on a 21 month schedule
Selection of expert	2
Selection of tool	2
Testing of tools	4
Knowledge engineering	6
Implementation	3
Validation	1
Evaluation	3
Maintenance	Ongoing

long as there is sufficient alternative advice or even scope for developmental experimental work.

3. Building an expert system for ion chromatography

3.1 Selection of the expert and domain

The selection of the (human) expert and the area of expertise is the most important stage in building an ES. The expert must be sufficiently recognized by his/her peers. More importantly, the expert must be willing to spend a considerable amount of time cooperating with the knowledge engineer to reveal the full depth of his/her knowledge. A busy analytical chemist may find it unsatisfying to rehearse at length the complex processes used to arrive at a decision. When building a system for the initial configuration several thousand cases must be assessed to bring out the required knowledge for even the sub-method of ion chromatography.

3.2 The knowledge base

The form of knowledge representation depends on facilities provided by the tools, and on the nature of the knowledge. Knowledge in ESs usually takes the form of *rules* and *facts*. The facts are the objects or concepts about which the expert reasons and rules are derived from this reasoning. An example fact for chromatography is that the equipment consists of a detector, an injector, a column, and a pump.

3.2.1 Rules

From the early days of AI it was recognized that the rules of inference of formal logic provided the best potential for the representation of knowledge. Rules in AI systems are used to represent the reasoning knowledge and employ functions allowed by procedural logic and predicate calculus. These include the following: IF, AND, NOT, OR, THEN, IMPLIES, EQUIVALENT,

TRUE/FALSE, IS A. Rules are supposed to represent the way in which the expert solves problems. For example, if a compound is non-volatile then liquid chromatography is preferred to gas chromatography, or if molecular weights are greater than 3000, then size-exclusion chromatography is applicable. These decisions can be represented as the following rules:

IF a compound is non-volatile
AND liquid chromatography is available
THEN use liquid chromatography

IF the molecular weight is more than 3000
AND liquid chromatography is available
AND size-exclusion chromatography is available
THEN use liquid chromatography
AND use size-exclusion chromatography

These examples show how apparently simple rules can become complex when they are fully implemented with all their exceptions and additions. They show the use of the 'IF condition(s) THEN conclusion(s)' process. Rules conclude new information from information already available. This new information can now be inserted into the knowledge base. The source of information for operation of a rule is the knowledge base, the user, or some external source. When building rules it is important to maintain easy legibility of the rule base. A simple piece of advice often offered to rule builders is that 'if the rule looks too big it probably is too big and should be split into smaller rules'. (This is itself an example of an IF ... THEN construct).

The term *meta-knowledge* is quite frequently used; this is knowledge about the knowledge base. For example, it may be the time and date of creation of a frame, or the name of the creator. It also offers the possibility to specialize global characteristics of the knowledge base locally. It can allow range restrictions for attribute values. For instance, the temperature range of a liquid chromatography oven is usually 35–150 °C and the wavelength range of a UV/VIS detector is 190–700 nm.

3.2.2 Frames

A powerful way of representing factual knowledge is through frame networks, as developed by Marvin Minsky (1975). A frame is a computer representation of an object or concept. It can also be defined as a schema or unit. A frame has attributes and values, which describe an object. Attributes can have a large number of values, and these are strings, numbers, or symbols. This concept is often defined by object/attribute/values triplicates. A typical frame for a liquid chromatography column is shown in Table 7.2. The column is represented as an object with various attributes and values. Any particular column can be defined by this frame by selecting the relevant values for each attribute. Other frames can be linked to this to form a network, for instance pre-columns, cartridges, or guard columns.

These links are formed by relations. In tools which support frame structures, two standard relations are normally provided, the IS A and INSTANCE relations. These links are illustrated in a small network shown in Table 7.3. Attribute values may then be inherited. These default values can be subse-

Table 7.2 The representation of an HPLC column as a frame

Object	Attributes	Allowed values
Column	Manufacturer	BDH Altex Phase Separations
	Stationary phase	ODS C2 PL–gel 50A
	Particle size	5 μm 10 μm
	Internal diameter	2 μm 4.6 μm
	Length	10 cm 25 cm

Table 7.3 Network of HPLC columns

Frame	Relation	Frame
Microbore column	IS A	Column
GPC column	IS A	Column
2 mm × 10 cm C8	IS A	Microbore column
4.6 mm × 10 cm ODS	IS A	Reverse–phase column

Table 7.4 Example of relations which can be user defined

Frame	Relation	Frame
Microbore columns	REQUIRE A	1 μL flow cell
Fast HPLC columns	REQUIRE A	2.5 μL flow cell
Detector	IS PART OF	HPLC system
Flow cell	IS PART OF	Detector

quently overridden by new values. Some tools allow the developer to define relations, which can provide extra powers of expression. An example from liquid chromatography is shown in Table 7.4.

Another common feature permits a procedural attachment to an attribute (or slot). This is known as an active value or a *demon*. The demon watches the attribute for a specified access to it. When this occurs the demon is acti-

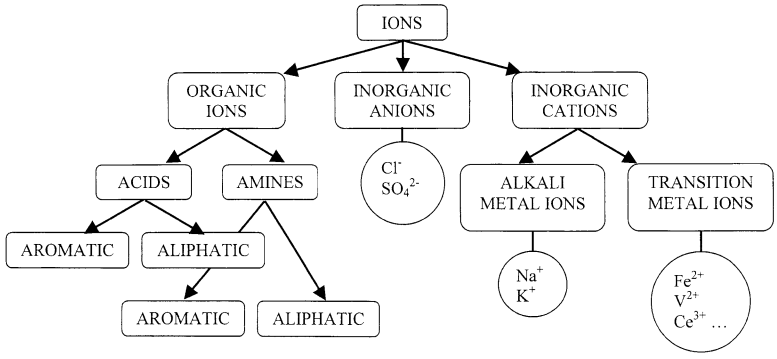

Fig. 7.4 A network of chemical compounds.

vated and some pre-defined action is taken. Typical uses of demons are protection against unauthorized access, update of an attribute, or the re-calculation of an object if any of its constituents are changed. For instance, if the internal diameter of the column is changed then this requires changes in flow-rate, flow cell, and injection volume. A demon could watch for a new input of the internal diameter and, when necessary, activate the required changes in flow-rate, etc.

From these examples it can be seen that a complete HPLC or any other analytical instrument can be represented using frames. Appropriate rules can then be developed to interact with a network of frames. Frames also allow inheritance through a parent–child structure, often referred to as specialization–generalization hierarchies. Figure 7.4 shows a network for chemical compounds. As the network progresses downwards, the classes become more specialized and each lower class inherits the properties of all the classes above it. For instance Fe^{3+} has all the properties of the transition metal ion class and this in turn has all the properties of cations. This can be a very useful representation as once an object is identified as part of one class, other information can be recognized about the object. Frames provide an efficient form of knowledge storage and prevent some unnecessary duplication of data. However, the example also highlights the problem of an essentially two-dimensional hierarchy. The aromatic/aliphatic choice could have been made one generation earlier applied to 'organic ions'. However the subsets acids and amines would then have been duplicated under each of 'aromatic' and 'aliphatic'.

3.2.3 Semantic networks

Semantic networks were first introduced by Ross Quillan (1968). The concept was proposed as a means to imitate the psychological model of human associative memory. Very simply, semantic networks are another way of representing knowledge in a frame system. They have two basic components, nodes and arcs. Nodes represent objects, concepts, or situations and arcs represent the relationships between them. Nodes are indicated by boxes or circles and arcs by arrows. The information described above in frames could

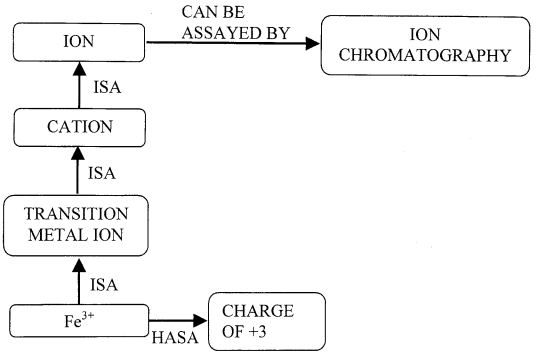

Fig. 7.5 An example semantic network for the analysis of Fe^{3+} by ion chromatography.

be represented in a semantic net where the information in Table 7.2 would be represented in nodes and the relations shown in Tables 7.3 and 7.4 would be arcs. A simple semantic network for the analysis of Fe^{3+} by ion chromatography is shown in Fig. 7.5. The advantage of these networks is the ease with which relationships can be viewed. It is a simple process to translate frames to semantic networks or vice-versa. Each object in the frame becomes a node in the semantic network and each slot (or relationship) becomes an arc. Thus Fe^{3+} inherits 'can be assayed by ion chromatography' through its membership of the class of 'transition metal ions', which in turn is inherited through membership of 'cations' and ultimately 'ions'.

3.3 The reasoning mechanism

The reasoning mechanism of the inference engine is the method by which the knowledge base is accessed to produce a conclusion. There are two main processes adopted in ESs, forward and backward chaining. These are comprehensively explained by Harman and King (1985).

Development tools usually offer one of these strategies, though some offer both. Examples of how these strategies perform can be demonstrated by the use of the following rules:

(a) $A \& B \Rightarrow D$
(b) $D \Rightarrow H$
(c) $D \& B \Rightarrow E$
(d) $F \& H \& E \Rightarrow X$
(e) $E \Rightarrow F$

These rules are read as follows: rule (a) means that if A and B are known then D can be concluded. Forward chaining begins with some facts, from either the consultation or the knowledge base. The rules run with this information to make any potential conclusions. In this example A and B are unknown. The first conclusion is from rule (a), i.e., conclude D. At this stage it is possible to

use either rule (b) or rule (c). A conflict resolution strategy is employed to determine which has priority. The control the developer has over this strategy is dependent on the tool being used. For this example it is assumed that concluding from single facts has precedence over multiple facts and so rule (c) take preference. Thus, the order of conclusions is as follows: conclude E, conclude H, conclude F, conclude X. At this point there are no more rules to be activated, so the result of the reasoning strategy is that D, E, H, F, and X can be concluded if A and B are known.

Backward chaining differs in that the goal is first set for the rules to conclude. For example, if the goal is set to conclude X, rule (d) shows that to prove X it is required to prove F, H, and A. Each of these then becomes a goal, so the firing of the rules now runs as follows:

to prove F prove E
to prove E prove B and D
to prove D prove A and B
to prove H prove D.

In conclusion it can be seen that X can be concluded if A and B are known. The classical example of a programming language that employs a forward chaining rule system is OPS5, and one that uses backward chaining is Prolog. Forward chaining is more appropriate for problems such as configuration tasks where there are a small number of starting states and a large number of finishing states (goals). Backward chaining is better for solving diagnostic problems where there are a small number of goals (faults) and a large number of starting conditions (causes). The larger development tools allow forward and backward chaining to be mixed as the problem dictates. The smaller tools may permit only one or the other.

Matt Ginsberg (1993) remarks that AI systems use knowledge to reduce problems of intelligent action to a search. Many tasks in AI can be phrased in terms of a search for the solution to the problem at hand. ESs require some mechanism to search the rule base for a suitable decision or conclusion. Numerous search techniques are possible, but many are susceptible to the problem of combinatorial explosion. If the size of the system is specified by a parameter n, the search space is typically 2^n or $n!$. Consider the case of finding an IC method that combines values for 8 features (eluent, detector, column, etc). Some of these attributes can have 40 possible values. A brute force search with no IC knowledge would require a lengthy search of 8^{40} (1×10^{36}) possible states. Various search methods have been proposed from the mathematics and computer science communities. Uninformed methods (known as *weak methods*) such as blind hill-climbing methods, genetic algorithms, or evolutionary strategies (Davis 1991) are less efficient than those that employ domain knowledge to reduce the search space and improve efficiency.

Chromatographers are not newcomers to the field of effective search. Many workers have evaluated methods to search for eluent combinations in what is often called solvent optimization strategies. SIMPLEX is a weak method that uses a hill climbing search method. Whereas the technique developed by Schoenmakers (1986) uses fewer experiments, each of these is

designed with a knowledge of chromatography and they provide a much more efficient search. In a similar manner AI has adopted a concept of knowledge based heuristics for constraining and directing search (Firebaugh 1989). These methods are known as control systems as they control the use of the rules. There are many search techniques that can be applied, including depth first or breadth first, these methods are embedded in the inference engine of the ES tool.

3.4 The interfaces

Two interfaces are part of an ES. The first is the developer's interface, which is important because it determines the ease and speed of building a system. The interface should be easy to use and contain features such as graphical representation to aid the development of networks. There should be comprehensive debugging and trace facilities that allow the examination of all the rules and the order of activation for a particular consultation of the ES. It should be easy to develop prototype systems, in order to build up a system in stages and to test each stage with a minimum of effort. It is also important to consider the qualifications and experience of the knowledge engineer. A competent software engineer could handle even the most complex of tools. On the other hand the chemist or analyst with little programming experience needs to select a simpler tool.

The second user interface is that for the eventual consultant of the software. More flexible tools allow the creation of interfaces by the developer, but others provide a fixed interface. User interfacing is not a particularly strong point of most tools. As the final interface can be complex with many features, such as explanation facilities, it must be carefully designed to avoid unduly cluttered screens or convoluted procedures.

3.5 Hardware and software

Any system is limited by the ES development software and hardware. Generally, a larger ES requires more investment in software and hardware. The investment should reflect the value of the final system. For instance, chromatographic applications are mostly confined to the PC environment, and although some chromatographers use workstation applications an ES written for this environment would not be widely used.

ES development tools have two major restrictions, they are memory hungry and slow. This always restricts the size of knowledge bases. It is often better to build an integration of small modules that can run with reasonable memory requirements and speed.

3.5.1 Development tools — conclusions

Table 7.3 shows a summary of some of the features of the development tools evaluated within the project 'Expert systems applied to Analytical Chemistry' (Goulder *et al.* 1988), which was funded as part of the European Strategic Program for Research in Information Technology.

Table 7.5 Comparison of development tools

Features	Shells	Semi–tools	Tools
Price ($)	100s	1000s	5000s
Knowledge representation	Average	Good	Excellent
Inferencing	One	Two	Two
Externals*	Poor	Good	Excellent
Numeric capability	Poor	Good	Excellent
Hardware	PC	Expanded PC Work station	
Examples	Delfi, Kes, Crystal	Goldworks, N expert, Xi Plus	ART, KEE, KC, Level 5 Object

'Externals' is the ability of the tool to connect with external routines, databases, or sources of data.

4. Applications of expert systems in analytical chemistry

4.1 Introduction

Expert systems have captured the interest of many chemists (Bridge 1990; Buydens and Schoenmakers 1993; Peris 1996) and indeed a chemical system, DENDRAL, was one of the first ESs. DENDRAL, still considered a benchmark system in AI, was developed in 1964 and has progressed through several versions since (Firebaugh 1989, p. 338). Indeed, the success of DENDRAL has been cited as the spur to the exceptional interest in ES in chemistry.

4.2 Dendral

Joshua Lederberg, a Nobel prize winner in genetics, devised a program, DENDRAL (**dendr**itic **al**gorithm), for enumerating all possible configurations of a set of atoms. To reduce the number of potential configurations produced by Lederberg's original system to a manageable number, Feigenbaum developed an expert system (known originally as heuristic DENDRAL but later just referred to as DENDRAL) that used mass spectral information and expertise. Feigenbaum describes the problem domain as follows:

'It was a problem which had all the elements of classical empirical induction. Here's an array of data that comes from a physical instrument, the mass spectrometer. Here's a set of primitive constructs out of which to compose a hypothesis about what organic compound is being analyzed. Here's a legal-move generator for generating all possible hypotheses. The problem is to find good ones out of all the set of possible ones, since in the general case, you don't want to generate all possible hypotheses. How do you find the good ones? And how do you employ knowledge of the world of chemistry, mass spectrometry, to constrain the set of alternatives, steering away from large sets of unfruitful ones? That was the framework (of the problem)'.

In the analysis of mass spectra, there is no numerical algorithm for mapping the mass spectrum to the structure of a compound. The expertise and experience of a trained mass spectrometrist is brought to bear on the problem. The task for Feigenbaum was to build an expert system with Lederberg's expertise. As with many ES projects it required the expert, Lederberg, to learn about computers and the knowledge engineer, Feigenbaum, to learn about chemistry. This project was the first to identify the problem of 'the knowledge acquisition bottleneck' discussed above. The team was also the first to identify problems with experts explicating their expertise It had become clear that in addition to the many rules of chemistry, chemists relied on a vast body of heuristic knowledge based on experience and simply guessing.

In 1970 a project was launched to develop rules automatically by examining examples of previously interpreted mass spectra. This was known as the Meta-DENDRAL project and it was an attempt to avoid the pitfalls of using human experts.

Although DENDRAL is held by the AI community as a seminal ES, it has never attracted a following in chemistry.

4.2.1 A description of the DENDRAL program

The input data to DENDRAL typically consists of the following information on the compound under study:

- The empirical chemical formula e.g., $C_8H_{16}O$
- The mass spectrum of the compound
- Nuclear Magnetic Resonance (NMR) spectroscopic information

There are three basic stages in the identification of structure, PLAN, GENERATE, and TEST.

The PLAN stage reduces the set of possible configuration of atoms by the constraints derived from the mass spectrum. The constraints are applied in two ways; the molecular fragments that must be included in the final structure and those that must not appear.

The GENERATE stage uses the constraints from the PLAN stage to generate all structures of the empirical formula containing the identified molecular fragments and removing those which should be excluded.

The TEST stage ranks the output of the GENERATE stage according to the quality of the fit between the hypothesized structure and the experimental one.

The following are some example rules:

IF the spectrum from the molecule has two peaks at masses XI and X2 such that:

X1 + X2 = M + 28 AND
X1 – 28 is a high peak AND
X2 – 28 is a high peak AND
at least one of X1 or X2 is high THEN

The molecule contains a ketone group

IF there is a high peak at mass 71 AND
There is a high peak at mass 43 AND
There is a high peak at mass 86 AND
There is a high peak at mass 58 THEN
There must be an N-PROPYL-KETONE3 structure

In the example given above, $C_8H_{16}O$, DENDRAL originally reduces the possible atom arrangements from 698 to about 40 ketones using the first rule and then arrives at the singular answer $CH_3CH_2(CO)CH_2CH_2CH_2CH_2CH_3$ using the second rule. If DENDRAL cannot arrive at a unique solution, it presents a list of possible structures together with a probability estimate of each one.

DENDRAL is written in LISP and is available as a commercial system in the USA. It claims to perform at the level of a chemistry PhD or above in its strictly defined domain. Perhaps the most notable feature of the DENDRAL project was that it was the first move away from the attempt to develop general problem solvers in AI. It marked the development of domain specific and knowledge-rich problem solvers. This was a major paradigm shift for the AI community.

4.3 ECAT

ECAT (**E**xpert **C**hromatography **A**ssistance **T**eam) was the first major attempt to develop an ES for the chromatography domain. It was developed by Varian Associates and the goal of the project was defined as follows (Bach *et al.* 1986):

'The goal of our project is to create a computer program that performs, at the human expert level, the tasks of designing, analyzing, optimizing, and trouble-shooting a high performance liquid chromatography (HPLC) separation method.'

The complete ES envisioned by the team was to provide for the following tasks:

1. To provide chemical information.
2. To choose between gas chromatography and liquid chromatography.
3. To specify the column, fluent constituents, and detector.
4. To decide on a sample clean-up procedure.
5. To optimize (or redesign) the separation.
6. To diagnose hardware problems

The program was to be implemented in ZETALISP and FRANZLISP in the UNIX environment. This project achieved only task 3 above and then only for a limited number of chemical samples. The team found that the extent of an expert's domain knowledge typically exceeds that which he or she realizes, and the knowledge engineering was non-trivial (Williams *et al.* 1989).

Consider task 1 of the goals of this project. This module was to have as its input the chemical name of the sample and perhaps its structure. The ES would then provide information on its chemical properties. An undergraduate degree can provide a chemist with only a limited ability to predict chemical

properties from a compound's structure. The chemist usually specializes in a limited domain of chemicals before real expertise can be built up for that domain of structures. Task 3 would also prove to be extremely taxing. Most chromatographers specialize in developing separation methods for a limited range of chemicals, so to build a general system would require many experts and thus a huge knowledge base. This project revealed the enormity of the task of building an ES for chromatography and also highlighted potential pitfalls in the knowledge engineering process. It never reached conclusion and was almost completely abandoned by the late 1980s.

4.4 ESCA

ESCA (**E**xpert **S**ystems Applied to **C**hemical **A**nalysis) was funded by ESPRIT (**E**uropean **S**trategic **P**rogramme for **R**esearch in **I**nformation **T**echnology). ESPRIT aimed to invest in both pre-development and product research projects in order to build and maintain a European competitive advantage in information technology. The ESCA project was of 3 years duration and represented 27 person-years of effort and approximately $3.5 million investment. The project began in 1987 and was completed in 1990.

This project began with much more general goals than either DENDRAL or ECAT, to purchase several ES development tools and to investigate their usefulness for the development of methods for HPLC (Mulholland 1992). Knowing the problems experienced by the ECAT team with their attempt to develop an ES for chromatography, the developers of ESCA spent the first two to four months defining a series of potential domains. Each domain tackled only one part of the method development process applied to a limited number of sample applications (Goulder *et al.* 1988). The four domains of expertise were as follows:

1. A best first guess at chromatographic conditions for central nervous system drugs (Hamoir *et al.* 1992).
2. Optimization criteria to evaluate chromatograms (Peeters *et al.* 1988).
3. Optimization of chromatographic parameters such as flow cell and column dimensions, flow-rate and the detector time constant (Schoenmakers and Dunand 1989).
4. Advice on the validation of chromatographic methods with regards to repeatability, reproducibility, and ruggedness testing (Mulholland *et al.* 1989*a*, 1989*b*)

A number of ES development tools and environments were evaluated. These systems were built using a number of ES development tools including KES, Goldworks, and N-Expert Object. Some of the systems were later re-implemented in a more conventional software environment (Pascal) in order to improve speed and memory requirements. The ESCA project then attempted to integrate some of these systems to test the feasibility of building large multi-functional systems for chromatography (Yuzhu *et al.* 1989; Mullholland *et al.* 1991; Hamoir *et al.* 1992; Buydens *et al.* 1993). The team concluded that effective ESs could be built to tackle problems in chromatography, provided

the domains were strictly defined. These systems could later be integrated to provide useful communication links. The overall conclusion was that ESs could not replace the analyst but they could provide tools to make analysts more efficient. The following list summarized the gains that could be expected from the introduction of a HPLC ES to a laboratory:

- Faster method development
- Better quality method development
- Faster analysis
- Distribution of senior scientists' knowledge to junior group members, thus increasing the team's efficiency
- Introduction of new knowledge from external sources to the laboratory
- Improved documentation
- Improved consistency
- Enhanced potential for the analyst

5. Ripple down rules

5.1 The philosophy of ripple down rules

The Ripple Down Rules (RDR) approach provides a number of solutions to the limitations of traditional ESs which rely on an empirical reductionist philosophy embedded in the physical symbols hypothesis. Compton argues against the absolute acceptance of the reductionist philosophy in AI and proposes a theory that knowledge is developed for the context in which it is being used. RDR allows for the creation of knowledge in context by the addition of rules in a tree structure within clearly defined contexts. A rule that is added to an RDR rule base is associated with the context of the case that caused its creation. It will not apply in other contexts. This process can run into the same problem of combinatory explosion in the physical symbols hypothesis.

Although Compton questions the validity of the reductionism philosophy, he did not abandon it for the development of RDR. He simply proposed a format for linking these atoms of knowledge that attempted to avoid the problems encountered with the physical symbol hypothesis. The flexibility of RDR, which allowed the addition either of *ad hoc* rules or for a process of conjecture and refutation, largely solves many of the maintenance problems of traditional ESs. To assume that an ES could be built in a finite time capturing even the majority of the expertise in a domain has been shown to be naive. RDR is a genuine attempt to move away from this simplistic philosophy of knowledge engineering.

5.2 Theory of ripple down rules

An RDR ES is based on a binary tree. Each node of the tree can be considered a rule with attached rule conditions and conclusion. There are two branches from each node, the true branch and the false branch. The tree is

consulted by a forward chaining mechanism. When a data set is presented to an RDR tree, the node is evaluated (to be either true or false) and causes the data set to be passed to the next appropriate branch, this process continues down to a terminal leaf. The conclusion of the last true is returned as the diagnosis of this data set. When an RDR produces an incorrect conclusion a new rule is added to the tree. The new rule is added as a branch of the last rule reached (whether or not it was true). It is added to the true branch if the data was evaluated to true, and to the false if the data was evaluated to false.

In Fig 7.6 each dot in the diagram indicates a rule, and each branch illustrates a possible result of evaluating the rule — a true evaluation leads to the right, while false evaluation leads to the left. An essential part of a practical RDR system is the maintenance of *cornerstone* cases. These cases are data sets that caused some change to the system. For each rule in a system, there is cornerstone case. There are several reasons why these are important. Trivially, they provide a chronicle of the development of the system for documentation purposes. More fundamentally however, they provide a context within which new rules can be created.

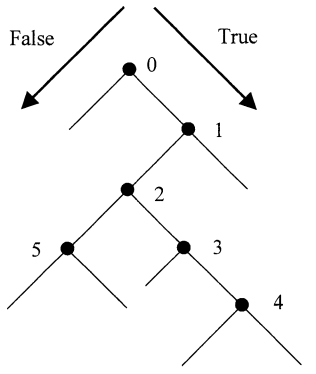

Fig. 7.6 A simple Ripple Down Rules tree.

This point is a crucial one in the effective implementations of RDR ESs. An expert will create a new rule when she/he does not agree with the conclusion produced by the system. The new rule she/he creates will differ in some way from the last set of true rule conditions, while satisfying data in the current case. In an effort to simplify new rules and make the new rule more concise, it is advantageous to provide the expert with a list of differences between the current case and the cornerstone case belonging to the last true rule. In other words the context for the new rule is provided by the case belonging to the last true rule.

In order to explain the process used to build an RDR tree, a simple example can be given. RDR trees are built using a selection of typical cases together with their conclusions. For this example, we will use the selection of a detector for an ion chromatography method. Table 7.6 shows the cases used in the development of the tree. These are purely speculative and are chosen simply to illustrate the process.

Table 7.6 Example cases with which to build an RDR tree

Case	Analyte	UV visibility	Post–column reactor	Suppressor	Detector applied for this case
1	Organic acid	yes	no	no	UV
2	Chloride	sample has both UV and non–UV absorbing solutes	no	yes	Conductivity
3	Transition metal	no	yes	no	UV
4	Iodide	yes	no	yes	Conductivity
5	Bromide	yes	no	no	Amperometric
6	Cyanide	no	no	no	Amperometric

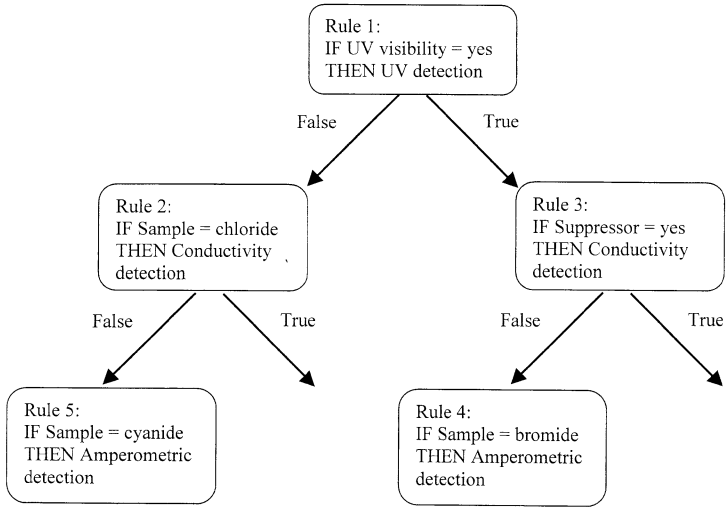

Fig. 7.7 An example RDR tree for the selection of an IC detector.

The first sample is an organic acid and is used as the cornerstone case to create Rule 1 in Fig. 7.7. This case can be used later to define the context in which Rule 1 was made. The second case in Table 7.6 is now presented to the RDR tree. As the case does not fall within the premise of rule 1 the system cannot provide a solution and another rule needs to be added to the system. To be consistent with a philosophy of knowledge in context the user is presented with a list of differences between the cornerstone case of Rule 1 and the current case. One or more of these differences can be selected as the premise(s) of Rule 2. As Rule 1 evaluated to false for this case the new rule is added on the false branch of that rule. Consider case 4; the RDR tree suggests the use of UV detection for this case as Rule 1 evaluates to true. However, this is an incorrect result. The user needs to add a new rule and is presented with a difference list from which to select new premises. This rule is added to the true branch of Rule 1. In this way the RDR tree begins to grow.

Unlike other ES methods, RDR trees do not distinguish between the development and maintenance stages. New rules can be added as required and because they are added in a controlled way, there is no need to validate the system with each new addition. Each new rule is embedded within the context of the cornerstone case and is not applicable outside this context.

5.3 Development cycle

The development cycle for ESs has been studied extensively (Gaines and Compton 1992), and is markedly different from ES technology. Maintenance and the evolution of an RDR system is a continual process of knowledge evolution, and so is indistinguishable from beta testing or prototyping. The utility of the ripple down rule approach has been demonstrated in large-scale applications. The 650 complex rules of a system for interpreting chemical pathology was reduced to some 550 simpler Ripple Down Rules, and the rate of

developing the rule base was increased from about 2 rules a day to 10 rules an hour. This successful implementation of RDR has lead to the development and implementation of a larger system at St Vincent's Hospital in Sydney. This system now contains some 1600 rules and is successfully being used in the day-to-day evaluation of reports from the Department of Chemical Pathology (Edwards *et al.* 1993).

5.4 Summary of the RDR method

RDR offer a novel approach to the development of an ES which could have long term advantages to the worth of an ES. The major advantages and disadvantages are summarized in the following sections.

5.4.1 Advantages of RDR

1. The major advantage of RDR lies in the simplicity of the technique. It automatically provides a consistent method of building an ES. The user does not require any programming skills, although knowledge of the front end software (Hypercard) allows the user opportunities to customize the system.
2. The philosophical underpinning of RDR is more attractive than that of the physical symbols hypothesis, however it has not removed all of the limitations of the reductionist philosophy. The technique has attempted to provide facilities that allow intelligence to be programmed by the creating rules within the context of their use. In this fashion it employs a belief that intelligence is a creative action.
3. The maintenance of the system can be carried out easily and in a controlled way that avoids the problems inherent in traditional ES maintenance.

5.4.2 Disadvantages of RDR

1. The technique is designed for classification problems with single conclusions being drawn from a prior list of options. However, many problems cannot be phrased as simple classifications. For example, selecting chromatographic conditions in ion chromatography is a configuration problem requiring a number of conclusions (e.g., the column, detector, eluent, etc) and these conclusions are dependent on each other. A case study of this system is presented later.
2. New rules are added within the context of the case that caused the change, so it is not possible to add rules which are not context specific. Consider the example tree described in Fig. 7.7. Rule 3 (which states that if a suppressor is available, use conductivity) was added as an exception to Rule 1. This rule is actually generally applicable, and it is true for many other contexts. It is an exception to Rule 5 and will also be an exception down the false branch of Rule 3. This means that the same knowledge has to be re-implemented at several places in the tree. The problem is alleviated if general rules are programmed near to the top of the tree, but there is no way of knowing in advance the most general cases with which to begin the RDR tree. Also, knowledge changes with time and new general knowledge will always be required.

This problem often exhibits itself when the RDR development is well under way and the tree is complex. A user can add a case which describes a new general rule only to find the next case follows a different path down the tree not reaching the relevant knowledge. This can be not only frustrating but can lead directly to the same problem of combinatory explosion experienced with traditional AI. Hence RDR trees can become very large and the problem of adding knowledge is not completely solved by the RDR format.

5.5 A RDR expert system for ion chromatography

The development of an ES in RDR was based on a database of all the significant published methods employing IC between 1980 and 1996 (Haddad and Jackson 1990). This amounts to over four thousand examples of IC. An example of a case in the database is shown in Table 7.7. Only the necessary information was extracted from the case and encoded as a series of attribute values, which were augmented by relevant information about the analyte: molecular weight, charge, if an acid the proticity and pK_a values, and the hydrophobicity.

Further attributes were automatically added to the case from an internal look-up table that described the nature of the solutes and special characteristics that may lead to the desired detection. Table 7.8 gives a list of the attributes that were used in the ES. The configuration problem was to fill in unknown attribute values for a given case. Usually these would be method attributes such as column, mobile phase, whether a suppressor is used, etc. However, if a laboratory had only one type of detector, for example, this would become a given attribute and the ES would have to work around this constraint to arrive at a suitable method.

An expert system using RDR was able to achieve 70–80 per cent agreement with a human expert when given a test set from the database. This was considered a good result as the database itself contained conflicting cases, and did not represent a statistically distributed set of cases.

Table 7.7 A typical case for IC from which the RDR expert system was built

Attributes	Values
Record	3968
Hardware	Dionex QIC
Column	Dionex AS–1 ion–exclusion, 250 × 2.0 mm ID
Packing	Cross–linked PS–DVB cation exchanger
Eluent	2.0 mM sulfuric acid; 0.8 ml/min
Solutes	Lactic (6.9), tartaric (8.0), malic (10.0), acetic (13.2)
Detection	Conductivity with various suppressor devices
Detection limit	2 ppm
Sample	White wine
Preparation	Dilution, filtration
Temperature	Ambient

Table 7.8 Attributes used by RDR in analysis of IC methods

Attribute mnemonic	Description	Number of values	Examples
Solute attributes			
APPLICAT	For what application was the method used?	14	environmental, pharmaceutical,
TYPE	The nature of the solute charge	3	cation, anion, neutral
PH_SOLUTE	Is solute acidic or basic?	3	acidic, basic, neutral
SOLUBLTY	Solubility of the solute	4	slightly ... very
CHARGE	Charge on solute ion	9	+4 ... –4
IONCLASS	Chemical nature of the solute	6	inorganic, organic
SUPPRESS	Is suppressor used?	2	yes, no
NO_SOLUT	Number of ions to be separated	3	1–5, 6–10, > 10
UV_ABSOR	Solute contains UV absorbing species?	3	both absorbing and non–absorbing, yes, no
HALIDES	Whether halides are to be assayed, and their UV absorbing properties	4	Uvhalide, none, non_Uvhalide, both
SULFATES	Sulfate or sulfite to be assayed?	2	yes, no
NITRATES	Nitrates or nitrites to be assayed?	2	yes, no
Method attributes			
MECHANSM	Chemical mechanism of the separation	5	ion exchange, ion interaction
POST_COLM	Is a post–column reaction used?	2	yes, no
pH	pH of mobile phase	4	acidic, basic, neutral, unspecified
MOBILE_P	Eluent composition	48	water, H_2SO_4, HCl
GRADIENT	Gradient elution used?	2	yes, no
COLUMN	Column type	7	neutral silica, cation exchanger resin, crown ether
DETECTOR	Detector type	18	conductivity, amperometry, refractive index

6. Conclusions

Artificial intelligence and expert systems, in particular, have been the subjects of much hype and media-built expectation. ESs apply *human* terminology to *non-human* things and use their own peculiar language of fuzzy logic, demons, rules, and frames. All this has had the effect of surrounding the field with an aura of mystery. It is interesting to speculate why ESs received so much attention in the 1970s and 1980s and why interest has waned in the 1990s. There were several reasons for the initial enthusiasm. First, ESs promised to make information technology and computer software more human and less algorithmic. This would allow computers to tackle a whole new area of human problems. The ability of ESs to imitate real experts also raised expectations that these problem solvers would be easier to use and to learn from than conventional software. Secondly, workers increasingly needed to access expertise from outside their own subject domain. Hence, there was a real need for expertise transfer and ESs promised to tackle this need. Finally, it is a recognized business maxim that a company's most valuable asset is its personnel. ESs could potentially allow this experience and skill to be preserved within a company if personnel retire or leave. Together, these reasons led to the investment of millions of dollars in this technology.

So why did this investment wane in the 1990s? Mainly due to the disappointing results achieved so far by the technology. Although many successful ESs were built and are still in use, the technology did not live up to its expected potential. Like much of AI, ESs failed to deliver the overblown promises of the early practitioners. The all-knowing HAL of Kubrick's 2001 was not going to be realized before the millennium. Indeed, it became obvious that the computer science researchers did not know how to start realizing such a system. The following is a list of some of the lacunae that became evident with ESs:

- ESs are time consuming and expensive to produce.
- ESs are not robust. In other words, they quickly failed outside their limited domains.
- ESs are hard to keep up to date. Most systems simply provided a snapshot of the expertise applied over the time of their creation and quickly became out of date.
- It was more difficult than originally thought to extract knowledge from experts.

The limitations and capabilities of ESs are now better known, and by adopting a more conservative strategy there have been sufficient successes to show that the technology is worthwhile.

6.1 ES in analytical chemistry

If DENDRAL is considered an archetype by the AI community, it must be admitted that it is not highly regarded by the analytical community for many of the reasons given above.

In chromatography, which has received the most attention, the early optimism, which assumed building a general chromatography system was simply a matter of mining knowledge from chromatographers and implementing it in rules, has given way to a more pragmatic stance. The most recent ESs have been built to tackle only defined domains, but despite the existence of a number of systems there has been no widespread use of any of these. One of the reasons for this appears to be that the ES are only applicable within narrow domains. Several other factors seem to influence the reluctance to use ES. Laboratories mostly have their own defined procedures and are reluctant to accept outside expertise. It is also difficult to customize an ES to overcome this problem. ESs are brittle, without specifically coded expertise they quickly fail when applied outside the specified domain for which they were developed. An ES captures only a snap-shot of an expert's knowledge over the time-frame of the development of the system. They cannot easily self learn and maintenance of ES is a non-trivial task.

References

Bach, R., Karnicky, J., and Abbott, S. (1986). An expert system for high performance liquid chromatography methods development. In *Artificial intelligence applications in chemistry*. (ed. T.A. Pierce and B.A. Horne), pp. 278–96. American Chemical Society, Washington, DC.

Bridge, T. P. (1990). *Chromatogr. Anal.*, **11**, 13.

Buydens, L. and Schoenmakers, P. (1993). *Intelligent software for chemical analysis*. Elsevier, Amsterdam.

Buydens, L., Schoenmakers, P., Maris, F., and Hindriks, H. (1993). Expert systems in chromatography. Results of the ESCA project. *Anal. Chim. Acta*, **272**, 41–51.

Davis L. (ed.) (1991). *Handbook of genetic algorithms*. Van Nostrand-Reinhold, London.

Edwards, G., Compton, P., Malor, R., Srinivasan, A., and Lazarus, L. (1993). PEIRS: A pathologist-maintained expert system for the interpretation of chemical pathology reports. *Pathology*, **25**, 27–34.

Firebaugh, M.W. (1989). *Artificial intelligence, a knowledge-based approach*. PWS Publishing Company, Boston.

Gaines, B. and Compton, P. (1992). Induction of Ripple Down Rules, in *AI '92 Proceedings of the 5th Australian joint conference on artificial intelligence*. pp. 349–54. World Scientific, Singapore.

Ginsberg, M. (1993). *Essentials of artificial intelligence*. Morgan Kaufman, New York.

Goulder, D., Blaffert, T., Blokland, A., Buydens, L., Chabra, A., Cleland, A., Dunand, N., Hamoir, T., Bourguignon, B., Massart, D.L., and Hindriks H. (1988). Model building for the prediction of initial chromatographic conditions in pharmaceutical analysis using reversed-phase liquid chromatography. *J. Chromatogr.*, **633**, 43–56.

Haddad, P.R. and Jackson, P.E. (1990). *Ion chromatography, principles and applications*. Journal of Chromatography Library, Elsevier, Amsterdam.

Hamoir, T., de Smet, M., Pyrins, H., Conti, P., van den Driessche, N., Massart, D.L., Maris, F., Hindriks, H., and Schoenmakers, P.J. (1992). Feasibility study for the con-

struction of an integrated expert system in high-performance liquid chromatography. *J. Chromatogr.*, **589**, 31–43.

Harman, P. and King, D. (1985). *Artificial intelligence in business expert systems*. Wiley, New York.

Maris, F. and Hindriks, R. (1993). Validation and evaluation of expert systems for HPLC method development — case studies. In *Intelligent software for chemical analysis*, (ed. L.M.C. Buydens, and P.J. Schoenmakers) pp. 153–223. Elsevier, Amsterdam.

Mulholland, M. (1992). Expert systems and their implementation in chromatography. In *Microcomputers and their application in biochemistry*, (ed. C. Bryce), pp. 243–65. Oxford University Press.

Mulholland, M., van Leeuwen, J.A., and Vandeginste, B.G.M. (1989*a*). An expert system for designing an intelligent spreadsheet for evaluation of precision of liquid chromatographic methods. *Anal. Chim. Acta*, **223**, 183–92.

Mulholland, M., Dunand, N., Cleland, A., van Leeuwen, J.A., and Vandeginste, B.G. M. (1989*b*). Expert system for method validation in chromatography. *J. Chromatogr.*, **485**, 283–96.

Mulholland, M., Walker, N., van Leeuwen, J.A., Buydens, L., Maris, F., Hindriks, H., and Schoenmakers, P.J. (1991). Expert systems for method development and validation in HPLC. *Mikrochim. Acta*. **2**, 493–503.

Peeters, A., Buydens, L., Massart, D.L., and Schoenmakers, P.J. (1988). An expert system for the selection of criteria for selectivity optimization in high-pressure liquid chromatography. *Chromatographia*, **26**, 101–9.

Peris M. (1996). *CRC Crit. Rev. in Anal. Chem.*, **26**, 219–37.

Quillan, R.M. (1968). *Semantic memory*. In *Semantic information processing*, (ed. M. Minsky). MIT Press, Cambridge, MA.

Schoenmakers, P.J. (1986). *Optimisation of chromatographic selectivity — a guide to method development*. Elsevier, Amsterdam.

Schoenmakers, P.J. and Dunand, N. (1989). Explanations and advice provided by an expert system for system optimization in high-performance liquid chromatography. *J. Chromatogr.*, **485**, 219–36.

Williams, S.S., Karnicky, J.F., and Excoffier, J.L. (1989). Expert system program for assistance in high-performance liquid chromatographic method development. *J. Chromatogr.*, **485**, 267–81.

Yuzhu, H., Peeters, A., Musch, G., and Massart, D.L. (1989). Integration of optimization methodology with expert systems. *Anal. Chim. Acta*, **223**, 1–17.

8 Automatic design of analog electrical circuits using genetic programming

John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane

1. Introduction

The design process entails creation of a complex structure to satisfy user-defined requirements. The design of analog electrical circuits is particularly challenging because it is generally viewed as requiring human intelligence and because it is a major activity of practicing analog electrical engineers.

Design of analog circuits begins with a high-level description of the circuit's desired behavior and entails creation of both the topology and the sizing of a satisfactory circuit. The topology comprises the gross number of components in the circuit, the type of each component (e.g., a resistor), and a list of all connections between the components. The sizing involves specifying the values (typically numerical) of each of the circuit's components.

Considerable progress has been made in automating the design of certain categories of purely digital circuits; however, the design of analog circuits and mixed analog–digital circuits has not proved as amenable to automation (Rutenbar 1993). Describing 'the analog dilemma,' Aaserud and Nielsen (1955) noted

> Analog designers are few and far between. In contrast to digital design, most of the analog circuits are still handcrafted by the experts or so-called 'zahs' of analog design. The design process is characterized by a combination of experience and intuition and requires a thorough knowledge of the process characteristics and the detailed specifications of the actual product.
>
> Analog circuit design is known to be a knowledge-intensive, multi-phase, iterative task, which usually stretches over a significant period of time and is performed by designers with a large portfolio of skills. It is therefore considered by many to be a form of art rather than a science.

There has been extensive previous work on the problem of circuit design using simulated annealing, artificial intelligence, and other techniques as outlined in Koza *et al.* (1997), including work using genetic algorithms

(Kruiskamp and Leenaerts 1995; Grimbleby 1995; Thompson 1996). However, there has previously been no general automated technique for synthesizing an analog electrical circuit from a high-level statement of the desired behavior of the circuit.

Once the user has specified the high-level design goals for an analog circuit, it would be ideal if an automated design process could create *both* the topology and the sizing of a circuit that satisfies the design goals. That is, it would be ideal to have an automated 'What You Want Is What You Get' ('WYWIWYG' — pronounced 'wow-eee-wig') process for analog circuit design.

This chapter presents a uniform approach to the automatic design of both the topology and sizing of analog electrical circuits. Section 2 presents design problems involving five prototypical analog circuits. Section 3 describes genetic programming. Section 4 describes the method by which genetic programming is applied to the problem of designing analog electrical circuits. Section 5 details required preparatory steps. Section 6 shows the results for the five problems. Section 7 cites other circuits that have been designed by genetic programming.

2. Five problems of analog design

In this chapter we apply genetic programming to a suite of five problems of analog circuit design. The circuits comprise a variety of types of components, including transistors, diodes, resistors, inductors, and capacitors. The circuits to be designed have varying numbers of inputs and outputs, and are as follows:

(1) Design a lowpass filter having a one-input, one-output composed of capacitors and inductors and that passes all frequencies below 1000 Hz and suppresses all frequencies above 2000 Hz.

(2) Design a tri-state frequency discriminator (source identification) circuit having one input and one output that is composed of resistors, capacitors, and inductors and that produces an output of 1/2 volt and 1 volt for incoming signals whose frequencies are within 10 per cent of 256 Hz and within 10 per cent of 2560 Hz, respectively, but produces an output of 0 volts otherwise.

(3) Design a computational circuit having one input and one output that is composed of transistors, diodes, resistors, and capacitors and that produces an output voltage equal to the square root of its input voltage.

(4) Design a time-optimal robot controller circuit having two inputs and one output that is composed of the above components and that navigates a constant-speed autonomous mobile robot with nonzero turning radius to an arbitrary destination in minimal time.

(5) Design an amplifier composed of the above components and that delivers amplification of 60 dB (i.e., 1000 to 1) with low distortion and low bias.

3. Genetic programming

Genetic programming is a domain-independent problem-solving approach in which computer programs are evolved to solve, or approximately solve, problems. Genetic programming is based on the Darwinian principle of reproduction and survival of the fittest and analogs of naturally occurring genetic operations such as *crossover* (*sexual recombination*) and *mutation*.

John Holland's pioneering *Adaptation in natural and artificial systems* (1975) described how an analog of the evolutionary process can be applied to solving mathematical problems and engineering optimization problems using what is now called the *genetic algorithm* (GA). The genetic algorithm attempts to find a good (or best) solution to the problem by genetically breeding a population of individuals over a series of generations. In the GA, each *individual* in the population represents a candidate solution to the given problem. The GA transforms a *population* (set) of individuals, each with an associated *fitness* value, into a new *generation* of the population using reproduction, crossover, and mutation.

Books on genetic algorithms include those that survey the entire field, such as Goldberg (1989), Michalewicz (1992), and Mitchell (1996) as well as others that specialize in particular areas, such as the application of genetic algorithms to robotics (Davidor 1990), financial applications (Bauer 1994), image segmentation (Bhanu and Lee 1994), parallelization (Stender 1993), and simulation and modeling (Stender *et al.* 1994), control and signal processing (Man *et al.* 1997), and engineering design (Gen and Chen 1997).

Genetic programming addresses one of the central goals of computer science, namely automatic programming. The goal of automatic programming is to create, in an automated way, a computer program that enables a computer to solve a problem. Paraphrasing Arthur Samuel (1959), the goal of automatic programming concerns,

How can computers be made to do what needs to be done, without being told exactly how to do it?

In genetic programming, the genetic algorithm operates on a population of computer programs of varying sizes and shapes (Koza 1992). Genetic programming starts with a primordial ooze of thousands or millions of randomly generated computer programs composed of the available programmatic ingredients and then applies the principles of animal husbandry to breed a new (and often improved) population of programs. The breeding is done in a domain-independent way using the Darwinian principle of survival of the fittest, an analog of the naturally occurring genetic operation of crossover (sexual recombination), and occasional mutation. The crossover operation is designed to create syntactically valid offspring programs (given closure amongst the set of programmatic ingredients). Genetic programming combines the expressive high-level symbolic representations of computer programs with the near-optimal efficiency of learning of Holland's genetic algorithm. A computer program that solves (or approximately solves) a given problem often emerges from this process. See also Koza and Rice 1992.

Genetic programming breeds computer programs to solve problems by executing the following three steps:

(1) Randomly create an initial population of individual computer programs.
(2) Iteratively perform the following sub-steps (called a *generation*) on the population of programs until the termination criterion has been satisfied:
    (a) Assign a fitness value to each individual program in the population using the fitness measure.
    (b) Create a new population of individual programs by applying the following three genetic operations. The genetic operations are applied to one or two individuals in the population selected with a probability based on fitness (with reselection allowed).
        (i) Reproduce an existing individual by copying it into the new population.
        (ii) Create two new individual programs from two existing parental individuals by genetically recombining sub-trees from each program using the crossover operation at randomly chosen crossover points in the parental individuals.
        (iii) Create a new individual from an existing parental individual by randomly mutating one randomly chosen sub-tree of the parental individual.
(3) Designate the individual computer program that is identified by the method of result designation (e.g., the *best-so-far* individual) as the result of the run of genetic programming. This result may represent a solution (or an approximate solution) to the problem.

Genetic programming has been applied to numerous problems in fields such as system identification, control, classification, design, optimization, and automatic programming. Between 1992 and 1997, over 800 papers were published on genetic programming.

Multi-part programs consisting of a main program and one or more reusable, parameterized, hierarchically-called sub-programs (called *automatically defined functions*) may also be evolved (Koza 1994a, b) An *automatically defined function (ADF)* is a function (i.e., sub-routine, sub-program, `DEFUN`, procedure) that is dynamically evolved during a run of genetic programming and which may be called by a calling program (or sub-program) that is concurrently being evolved. When automatically defined functions are being used, a program in the population consists of a hierarchy of one (or more) *reusable* function-defining branches (i.e., automatically defined functions) along with a main result-producing branch. Typically, the automatically defined functions possess one or more dummy arguments (formal parameters) and are reused with different instantiations of these dummy arguments. During a run, genetic programming evolves different sub-programs in the function-defining branches of the overall program, different main programs in the result-producing branch, different instantiations of the dummy arguments of the automatically defined functions in the function-defining branches, and different hierarchical references between the branches.

Architecture-altering operations enhance genetic programming with automatically defined functions by providing a way to automatically determine

the number of such sub-programs, the number of arguments that each sub-program possesses, and the nature of the hierarchical references, if any, among such sub-programs (Koza 1995). These operations include branch duplication, argument duplication, branch creation, argument creation, branch deletion, and argument deletion. The architecture-altering operations are motivated by the naturally occurring mechanism of gene duplication that creates new proteins (and hence new structures and new behaviors in living things) (Ohno 1970).

Recent research on genetic programming is described in books (Banzhaf *et al.* 1997), edited collections of papers (Kinnear 1994, Angeline and Kinnear 1996), conference proceedings (Koza *et al.* 1996, 1997), and the World Wide Web (www.genetic-programming.org).

Before applying genetic programming to a problem, the user must perform five major preparatory steps. These five steps involve determining:

(1) the set of terminals,
(2) the set of primitive functions,
(3) the fitness measure,
(4) the parameters for controlling the run, and
(5) the method for designating a result and the criterion for terminating a run.

The first major step in preparing to use genetic programming is to identify the set of terminals. The terminals can be viewed as the inputs to the as-yet-undiscovered computer program. The set of terminals (along with the set of functions) are the ingredients from which genetic programming attempts to construct a computer program to solve, or approximately solve, the problem.

The second major step in preparing to use genetic programming is to identify the set of functions that are to be used to generate the mathematical expression that attempts to fit the given finite sample of data. Each computer program (i.e., parse tree, mathematical expression, LISP S-expression) is a composition of functions from the function set F and terminals from the terminal set T. Each of the functions in the function set should be able to accept, as its arguments, any value and data type that may possibly be returned by any function in the function set and any value and data type that may possibly be assumed by any terminal in the terminal set. That is, the function set and terminal set selected should have the closure property so that any possible composition of functions and terminals produces a valid executable computer program. For example, a run of genetic programming will typically employ a protected version of division (returning an arbitrary value such as zero when division by zero is attempted).

The evolutionary process is driven by the *fitness measure*. Each individual computer program in the population is executed and then evaluated, using the fitness measure, to determine how well it performs in the particular problem environment. The nature of the fitness measure varies with the problem. For many problems, fitness is naturally measured by the discrepancy between the result produced by an individual candidate program and the desired result. The closer this error is to zero, the better the program. In a problem of optimal control, the fitness of a computer program may be the

amount of time (or fuel, or money, etc.) it takes to bring the system to a desired target state. The smaller the amount, the better. If one is trying to recognize patterns or classify objects into categories, the fitness of a particular program may be measured by accuracy or correlation. For electronic circuit design problems, the fitness measure may involve how closely the circuit's performance (say, in the frequency or time domain) satisfies user-specified design requirements. If one is trying to evolve a good randomizer, the fitness might be measured by means of entropy, satisfaction of the gap test, satisfaction of the run test, or some combination of these factors. For some problems, it may be appropriate to use a multiobjective fitness measure incorporating a combination of factors such as correctness, parsimony (smallness of the evolved program), efficiency (of execution), power consumption (for an electrical circuit), or manufacturing cost (for an electrical circuit).

The primary parameters for controlling a run of genetic programming are the population size, M, and the maximum number of generations to be run, G.

Each run of genetic programming requires specification of a *termination criterion* for deciding when to terminate a run and a method of *result designation*. One frequently used method of result designation for a run is to designate the best individual obtained in any generation of the population during the run (i.e., the *best-so-far individual*) as the result of the run.

In genetic programming, populations of thousands or millions of computer programs are genetically bred for dozens, hundreds, or thousands of generations. This breeding is done using the Darwinian principle of survival and reproduction of the fittest along with a genetic crossover operation appropriate for mating computer programs. A computer program that solves (or approximately solves) a given problem often emerges from this combination of Darwinian natural selection and genetic operations.

Genetic programming starts with an initial population (generation 0) of randomly generated computer programs composed of the given primitive functions and terminals. Typically, the size of each program is limited, for practical reasons, to a certain maximum number of points (i.e. total number of functions and terminals) or a maximum depth (of the program tree). The creation of this initial random population is, in effect, a blind random parallel search of the search space of the problem represented as computer programs.

Typically, each computer program in the population is run over a number of different *fitness cases* so that its fitness is measured as a sum of an average over a variety of representative different situations. These fitness cases sometimes represent a sampling of different values of an independent variable or a sampling of different initial conditions of a system. For example, the fitness of an individual computer program in the population may be measured in terms of the sum of the absolute value of the differences between the output produced by the program and the correct answer to the problem (i.e., the Minkowski distance) or the square root of the sum of the squares (i.e., Euclidean distance). These sums are taken over a sampling of different inputs (fitness cases) to the program. The fitness cases may be chosen at random or may be chosen in some structured way (e.g., at regular intervals or over a regular grid). It is also common for fitness cases to represent initial conditions of a system (as in a control problem). In economic

forecasting problems, the fitness cases may be the daily closing price of some financial instrument.

The computer programs in generation 0 of a run of genetic programming will almost always have exceedingly poor fitness. Nonetheless, some individuals in the population will turn out to be somewhat more fit than others. These differences in performance are then exploited. The Darwinian principle of reproduction and survival of the fittest and the genetic operation of crossover are used to create a new offspring population of individual computer programs from the current population of programs.

The reproduction operation involves selecting a computer program from the current population of programs based on fitness (i.e., the better the fitness, the more likely the individual is to be selected) and allowing it to survive by copying it into the new population.

The crossover operation creates new offspring computer programs from two parental programs selected based on fitness. The parental programs in genetic programming are typically of different sizes and shapes. The offspring programs are composed of sub-expressions (sub-trees, sub-programs, sub-routines, building blocks) from their parents. These offspring programs are typically of different sizes and shapes than their parents.

For example, consider the following computer program (presented here as a LISP S-expression):

```
(+ (* 0.234 Z) (- X 0.789)),
```

which we would ordinarily write as

```
0.234 Z + X-0.789
```

This program takes two inputs (X and Z) and produces a floating point output.

Also, consider a second program:

```
(* (* Z Y) (+ Y (* 0.314 Z))).
```

One crossover point is randomly and independently chosen in each parent. Suppose that the crossover points are the * in the first parent and the + in the second parent. These two crossover fragments correspond to the underlined sub-programs (sub-lists) in the two parental computer programs.

The two offspring resulting from crossover are as follows:

```
(+ (+ Y (* 0.314 Z)) (- X 0.789))
(* (* Z Y) (* 0.234 Z)).
```

Thus, crossover creates new computer programs using parts of existing parental programs. Because entire sub-trees are swapped, the crossover operation always produces syntactically and semantically valid programs as offspring regardless of the choice of the two crossover points. Because programs are selected to participate in the crossover operation with a probability based on fitness, crossover allocates future trials to regions of the search space whose programs contains parts from promising programs.

The mutation operation creates an offspring computer program from parental programs selected based on fitness. One crossover point is randomly

and independently chosen and the sub-tree occurring at that point is deleted. Then, a new sub-tree is grown at that point using the same growth procedure as was originally used to create the initial random population.

After genetic operations are performed on the current population, the population of offspring (i.e., the new generation) replaces the old population (i.e., the old generation). Each individual in the new population of programs is then measured for fitness, and the process is repeated over many generations.

The hierarchical character of the computer programs that are produced is an important feature of genetic programming. The results of genetic programming are inherently hierarchical. In many cases the results produced by genetic programming are default hierarchies, prioritized hierarchies of tasks, or hierarchies in which one behavior subsumes or suppresses another.

The dynamic variability of the computer programs that are developed along the way to a solution is also an important feature of genetic programming. It is often difficult and unnatural to try to specify or restrict the size and shape of the eventual solution in advance. Moreover, advance specification or restriction of the size and shape of the solution to a problem narrows the window by which the system views the world and might well preclude finding the solution to the problem at all.

Another important feature of genetic programming is the absence or relatively minor role of preprocessing of inputs and postprocessing of outputs. The inputs, intermediate results, and outputs are typically expressed directly in terms of the natural terminology of the problem domain. The programs produced by genetic programming consist of functions that are natural for the problem domain. The postprocessing of the output of a program, if any, is done by a *wrapper (output interface)*.

Finally, another important feature of genetic programming is that the structures undergoing adaptation in genetic programming are active. They are not passive encodings (i.e., chromosomes) of the solution to the problem. Instead, given a computer on which to run, the structures in genetic programming are active structures that are capable of being executed in their current form.

Automated programming requires some hierarchical mechanism to exploit, *by reuse* and *parameterization*, the regularities, symmetries, homogeneities, similarities, patterns, and modularities inherent in problem environments. Sub-routines do this in ordinary computer programs.

Automatically defined functions can be implemented within the context of genetic programming by establishing a constrained syntactic structure for the individual programs in the population. Each multi-part program in the population contains one (or more) function-defining branches and one (or more) main result-producing branches. The result-producing branch usually has the ability to call one or more of the automatically defined functions. A function-defining branch may have the ability to refer hierarchically to other already-defined automatically defined functions.

Genetic programming evolves a population of programs, each consisting of an automatically defined function in the function-defining branch and a result-producing branch. The structures of both the function-defining

branches and the result-producing branch are determined by the combined effect, over many generations, of the selective pressure exerted by the fitness measure and by the effects of the operations of Darwinian fitness-based reproduction and crossover. The function defined by the function-defining branch is available for use by the result-producing branch. Whether or not the defined function will be actually called is not predetermined, but instead, determined by the evolutionary process.

Since each individual program in the population of this example consists of function-defining branch(es) and result-producing branch(es), the initial random generation must be created so that every individual program in the population has this particular constrained syntactic structure. Since a constrained syntactic structure is involved, crossover must be performed so as to preserve this syntactic structure in all offspring.

Genetic programming with automatically defined functions has been shown to be capable of solving numerous problems (Koza 1994*a*). More importantly, the evidence so far indicates that, for many problems, genetic programming requires less computational effort (i.e., fewer fitness evaluations to yield a solution with, say, a 99 per cent probability) with automatically defined functions than without them (provided the difficulty of the problem is above a certain relatively low break-even point). Also, genetic programming usually yields solutions with smaller average overall size with automatically defined functions than without them (provided, again, that the problem is not too simple). That is, both learning efficiency and parsimony appear to be properties of genetic programming with automatically defined functions.

Moreover, there is evidence that genetic programming with automatically defined functions is scalable. For several problems for which a progression of scaled-up versions was studied, the computational effort increases as a function of problem size at a *slower rate* with automatically defined functions than without them. Also, the average size of solutions similarly increases as a function of problem size at a *slower rate* with automatically defined functions than without them. This observed scalability results from the profitable reuse of hierarchically callable, parameterized sub-programs within the overall program.

When single-part programs are involved, genetic programming automatically determines the size and shape of the solution (i.e., the size and shape of the program tree) as well as the sequence of work-performing primitive functions that can solve the problem. However, when multi-part programs and automatically defined functions are being used, the question arises as to how to determine the architecture of the programs that are being evolved. The *architecture* of a multi-part program consists of the number of function-defining branches (automatically defined functions) and the number of arguments (if any) possessed by each function-defining branch. The architecture may be specified by the user, may be evolved using evolutionary selection of the architecture (Koza 1994*a*), or may be evolved using evolutionary selection of the architecture (Koza 1994a), or may be evolved using architecture-altering operations (Koza 1995).

4. Design by genetic programming

Genetic programming can be applied to circuits if a mapping is established between the program trees (rooted, point-labeled trees (acyclic graphs) with ordered branches) used in genetic programming and the line-labeled cyclic graphs germane to electrical circuits. The principles of developmental biology, the creative work of Kitano (1990) on using genetic algorithms to evolve neural networks, and the innovative work of Gruau (1992) on using genetic programming to evolve neural networks provide motivation for mapping trees into circuits by means of a growth process that begins with an embryo. For circuits, the embryo typically includes fixed wires that connect the inputs and outputs of the particular circuit being designed and certain fixed components (such as source and load resistors). The embryo also contains modifiable wires. Until these wires are modified, the circuit does not produce interesting output. An electrical circuit is developed by progressively applying the functions in a circuit-constructing program tree to the modifiable wires of the embryo (and, during the developmental process, to new components and modifiable wires). See also Brave 1996.

The functions in the circuit-constructing program trees are divided into four categories: (1) topology-modifying functions that alter the circuit topology, (2) component-creating functions that insert components into the circuit, (3) arithmetic-performing functions that appear in sub-trees as argument(s) to the component-creating functions and specify the numerical value of the component, and (4) automatically defined functions that appear in the function-defining branches and potentially enable certain substructures of the circuit to be reused (with parameterization).

Each branch of the program tree is created in accordance with a constrained syntactic structure. Branches are composed of construction-continuing sub-trees that continue the developmental process and arithmetic-performing sub-trees that determine the numerical value of components. Topology-modifying functions have one or more construction-continuing sub-trees, but no arithmetic-performing sub-tree. Component-creating functions have one or more construction-continuing sub-trees and typically have one arithmetic-performing sub-tree. This constrained syntactic structure is preserved using structure-preserving crossover with point typing (see Koza 1994*a*).

4.1 The embryonic circuit

An electrical circuit is created by executing a circuit-constructing program tree that contains various component-creating and topology-modifying functions. Each tree in the population creates one circuit. The specific embryo used depends on the number of inputs and outputs.

Figure 8.1 shows a one-input, one-output embryonic circuit in which VSOURCE is the input signal and VOUT is the output signal (the probe point). The circuit is driven by an incoming alternating circuit source VSOURCE. There is a fixed load resistor RLOAD and a fixed source resistor RSOURCE in the embryo. In addition to the fixed components, there is a modifiable wire Z0 between nodes 2 and 3. All development originates from this modifiable wire.

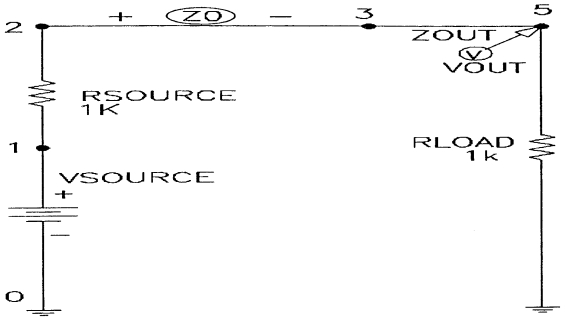

Fig. 8.1 One-input, one-output embryo.

4.2 Component-creating functions

Each program tree contains component-creating functions and topology-modifying functions. The component-creating functions insert a component into the developing circuit and assign component value(s) to the component.

Each component-creating function has a writing head that points to an associated highlighted component in the developing circuit and modifies that component in a specified manner. The construction-continuing sub-tree of each component-creating function points to a successor function or terminal in the circuit-constructing program tree.

The arithmetic-performing sub-tree of a component-creating function consists of a composition of arithmetic functions (addition and subtraction) and random constants (in the range –1.000 to +1.000). The arithmetic-performing sub-tree specifies the numerical value of a component by returning a floating-point value that is interpreted on a logarithmic scale as the value for the component in a range of 10 orders of magnitude (using a unit of measure that is appropriate for the particular type of component).

The two-argument resistor-creating R function causes the highlighted component to be changed into a resistor. The value of the resistor in kilo Ohms is specified by its arithmetic-performing sub-tree.

Figure 8.2 shows a modifiable wire Z0 connecting nodes 1 and 2 of a partial circuit containing four capacitors (C2, C3, C4, and C5). The circle indicates that Z0 has a writing head (i.e., is the highlighted component and that Z0 is subject to subsequent modification).

Figure 8.3 shows the result of applying the R function to the modifiable wire Z0 of Fig. 8.2. The circle indicates that the newly created R1 has a writing head so that R1 remains subject to subsequent modification.

Similarly, the two-argument capacitor-creating C function causes the highlighted component to be changed into a capacitor whose value in micro Farads is specified by its arithmetic-performing sub-trees.

The one-argument Q_ D_PNP diode-creating function causes a diode to be inserted in lieu of the highlighted component. This function has only one argument because there is no numerical value associated with a diode and thus no arithmetic-performing sub-tree. In practice, the diode is implemented here using a pnp transistor whose collector and base are connected to each

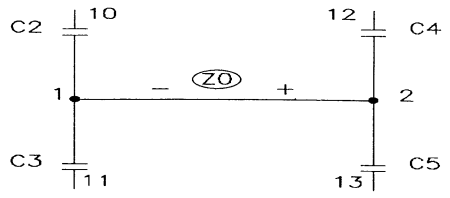

Fig. 8.2 Modifiable wire ZO.

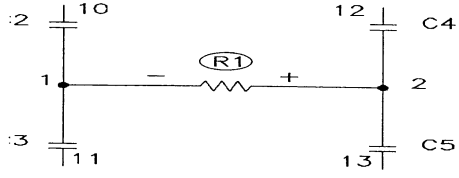

Fig. 8.3 Result of applying the R function.

other. The Q_D_NPN function inserts a diode using an npn transistor in a similar manner.

There are also six one-argument transistor-creating functions (Q_POS_COLL_NPN, Q_GND_EMIT_NPN, Q_NEG_EMIT_ NPN, Q_GND_EMIT_PNP, Q_POS_EMIT_PNP, Q_NEG_COLL_PNP) that insert a bipolar junction transistor in lieu of the highlighted component and that directly connect the collector or emitter of the newly created transistor to a fixed point of the circuit (the positive power supply, ground, or the negative power supply). For example, the Q_POS_COLL_NPN function inserts a bipolar junction transistor whose collector is connected to the positive power supply.

Each of the functions in the family of six different three-argument transistor-creating Q_3_NPN functions causes an npn bipolar junction transistor to be inserted in place of the highlighted component and one of the nodes to which the highlighted component is connected. The Q_3_NPN function creates five new nodes and three modifiable wires. There is no writing head on the new transistor, but there is a writing head on each of the three new modifiable wires. There are 12 members (called Q_3_NPN0, ..., Q_3_NPN11) in this family of functions because there are two choices of nodes (1 and 2) to be bifurcated and then there are six ways of attaching the transistor's base, collector, and emitter after the bifurcation. Similarly the family of 12 Q_3_PNP functions causes a pnp bipolar junction transistor to be inserted.

Figure 8.4 shows the result of applying the Q_3_NPN0 function, thereby creating transistor Q6 in lieu of the resistor R1 of Fig. 8.3.

4.3 Topology-modifying functions

Each topology-modifying function in a program tree points to an associated highlighted component and modifies the topology of the developing circuit.

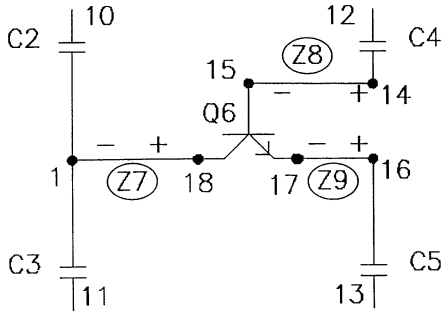

Fig. 8.4 Result of applying transistor-creating Q_3_NPNO function to resistor R1 of Fig. 8.3 transforming it into transistor Q6.

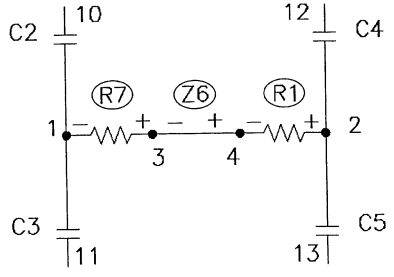

Fig. 8.5 Result after applying the SERIES function to resistor R1 of Fig. 8.3, thereby transforming it into resistors R7 and R1 and wire Z6.

The three-argument SERIES division function creates a series composition of the highlighted component (with a writing head), a copy of it (with a writing head), one new modifiable wire (with a writing head), and two new nodes. Fig. 8.5 illustrates the result of applying the SERIES division function to resistor R1 from Fig. 8.3.

The four-argument PARALLEL0 parallel division function creates a parallel composition consisting of the original highlighted component (with a writing head), a copy of it (with a writing head), two new modifiable wires (each with a writing head), and two new nodes. Figure 8.6 shows the result of applying PARALLEL0 to the resistor R1 from Fig. 8.3. The one-argument polarity-reversing FLIP function reverses the polarity of the highlighted component.

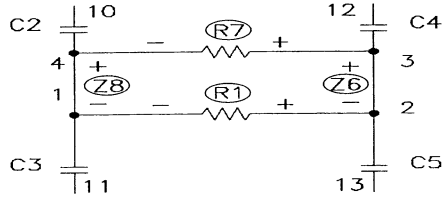

Fig. 8.6 Result of the PARALLEL0 function.

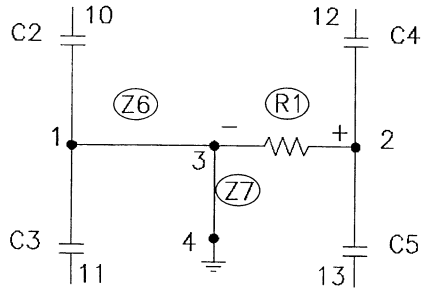

Fig. 8.7 Result of applying the T_GND_O function to resistor R1 of Fig. 8.3, thereby creating a connection to ground.

There are six three-argument functions (T_GND_0, T_GND_1, T_POS_0, T_POS_1, T_NEG_0, T_NEG_1) that insert two new nodes and two new modifiable wires, and then make a connection to ground, positive power supply, or negative power supply, respectively. Figure 8.7 shows the T_GND_0 function connecting resistor R1 of Fig. 8.3 to ground.

There are two three-argument functions (PAIR_CONNECT_0 and PAIR_CONNECT_1) that enable distant parts of a circuit to be connected together. The first PAIR_CONNECT to occur in the development of a circuit creates two new wires, two new nodes, and one temporary port. The next PAIR_CONNECT creates two new wires and one new node, connects the temporary port to the end of one of these new wires, and then removes the temporary port.

The one-argument NOOP function has no effect on the highlighted component; however, it delays activity on the developmental path on which it appears in relation to other developmental paths in the overall program tree.

The zero-argument END function causes the highlighted component to lose its writing head, thereby ending that particular developmental path.

The zero-argument SAFE_CUT function causes the highlighted component to be removed from the circuit provided that the degree of the nodes at both ends of the highlighted component is three (i.e., no dangling components or wires are created).

5. Preparatory steps

Before applying genetic programming to a problem of circuit design, seven major preparatory steps are required: (1) identify the suitable embryonic circuit, (2) determine the architecture of the overall circuit-constructing program trees, (3) identify the terminals of the program trees, (4) identify the primitive functions contained in these program trees, (5) create the fitness measure, (6) choose parameters, and (7) determine the termination criterion and method of result designation.

5.1 Embryonic circuit

The embryonic circuit used on a particular problem depends on the circuit's number of inputs and outputs .

For example, in the robot controller circuit, the circuit has two inputs, VSOURCE1 and VSOURCE2, not just one. Moreover, each input needs its own separate source resistor (RSOURCE1 and RSOURCE2). Consequently, the embryo has three modifiable wires Z0, Z1, and Z2 in order to provide full connectivity between the two inputs and the one output. All development then originates from these three modifiable wires.

There is often considerable flexibility in choosing the embryonic circuit. For example, an embryo with two modifiable wires (Z0 and Z1) was used for the lowpass filter. In some problems, such as the amplifier, the embryo contains additional fixed components because of additional problem-specific functionality of the harness (as described in Koza, Bennett, Andre, and Keane 1997).

5.2 Program architecture

Since there is one result-producing branch in the program tree for each modifiable wire in the embryo, the architecture of each circuit-constructing program tree depends on the embryonic circuit. One result-producing branch was used for the frequency discriminator and the computational circuit; two were used for the lowpass filter problem; and three were used for the robot controller and amplifier. The architecture of each circuit-constructing program tree also depends on the use, if any, of automatically defined functions. Automatically defined functions and architecture-altering operations were used in the frequency discriminator, robot controller, and amplifier. For these problems, each program in the initial population of programs had a uniform architecture with no automatically defined functions. In later generations, the number of automatically defined functions, if any, emerged as a consequence of the architecture-altering operations.

5.3 Function and terminal sets

The function set for each design problem depended on the type of electrical components that were used to construct the circuit. Capacitors, diodes, and transistors were used for the computational circuit, the robot controller, and the amplifier. Resistors (in addition to inductors and capacitors) were used for the frequency discriminator. When transistors were used, functions to provide connectivity to the positive and negative power supplies were also included.

For the computational circuit, the robot controller, and the amplifier, the function set, Fccs-initial for each construction-continuing sub-tree was

Fccs-initial = {R, C, SERIES, PARALLEL0, PARALLEL1, FLIP, NOOP, T_GND_0, T_GND_1, T_POS_0, T_POS_1, T_NEG_0, T_NEG_1, PAIR_CONNECT_0, PAIR_CONNECT_1, Q_D_NPN, Q_D_PNP, Q_3_NPN0, ..., Q_3_NPN11, Q_3_PNP0, ..., Q_3_PNP11, Q_POS_COLL_NPN, Q_GND_EMIT_NPN, Q_NEG_EMIT_NPN, Q_GND_EMIT_PNP, Q_ POS_EMIT_PNP, Q_NEG_COLL_PNP}.

For the npn transistors, the Q2N3904 model was used. For pnp transistors, the Q2N3906 model was used.

The initial terminal set, Tccs-initial, for each construction-continuing sub-tree was Tccs-initial = {END, SAFE_CUT}.

The initial terminal set, Taps-initial, for each arithmetic-performing sub-tree consisted of Taps-initial = {←},
where ← represents floating-point random constants from −1.0 to +1.0.

The function set, Faps, for each arithmetic-performing sub-tree was,

$$\text{Faps} = \{+, -\}.$$

The terminal and function sets were identical for all result-producing branches for a particular problem.

For the lowpass filter and frequency discriminator, there was no need for functions to provide connectivity to the positive and negative power supplies.

For the frequency discriminator, the robot controller, and the amplifier, the architecture-altering operations were used and the set of potential new functions, Fpotential, was

$$\text{Fpotential} = \{\text{ADF0}, \text{ADF1}, \ldots\}.$$

The set of potential new terminals, Tpotential, for the automatically defined functions was

$$\text{Tpotential} = \{\text{ARG0}\}.$$

The architecture-altering operations change the function set, Fccs for each construction-continuing sub-tree of all three result-producing branches and the function-defining branches, so that

$$\text{Fccs} = \text{Fccs-initial} \approx \text{Fpotential}.$$

The architecture-altering operations generally change the terminal set for automatically defined functions, Taps-adf, for each arithmetic-performing sub-tree, so that

$$\text{Taps-adf} = \text{Taps-initial} \approx \text{Tpotential}.$$

5.4 Fitness measure

The fitness measure varies for each problem. The high-level statement of desired circuit behavior is translated into a well-defined measurable quantity that can be used by genetic programming to guide the evolutionary process. The evaluation of each individual circuit-constructing program tree in the population begins with its execution. This execution progressively applies the functions in each program tree to an embryonic circuit, thereby creating a fully developed circuit. A netlist is created that identifies each component of the developed circuit, the nodes to which each component is connected, and the value of each component. The netlist becomes the input to the 217,000-line SPICE (Simulation Program with Integrated Circuit Emphasis) simulation program Quarles *et al.* 1994. SPICE then determines the behavior of the circuit. It was necessary to make considerable modifications in

SPICE so that it could run as a submodule within the genetic programming system.

5.4.1 Lowpass filter

A simple *filter* is a one-input, one-output electronic circuit that receives a signal as its input and passes the frequency components of the incoming signal that lie in a specified range (called the *passband*) while suppressing the frequency components that lie in all other frequency ranges (the *stopband*).

The desired lowpass LC filter should have a passband below 1000 Hz and a stopband above 2000 Hz. The circuit is driven by an incoming AC voltage source with a 2 volt amplitude. If the source (internal) resistance RSOURCE and the load resistance RLOAD in the embryonic circuit are each 1 kilo Ohm, the incoming 2 volt signal is divided in half.

The *attenuation* of the filter is defined in terms of the output signal relative to the reference voltage (half of 2 volt here). A *decibel* is a unitless measure of relative voltage that is defined as 20 times the common (base 10) logarithm of the ratio between the voltage at a particular probe point and a reference voltage.

In this problem, a voltage in the passband of exactly 1 volt and a voltage in the stopband of exactly 0 volts is regarded as ideal. The (preferably small) variation within the passband is called the *passband ripple*. Similarly, the incoming signal is never fully reduced to zero in the stopband of an actual filer. The (preferably small) variation within the stopband is called the *stopband ripple*. A voltage in the passband of between 970 millivolts and 1 volt (i.e., a passband ripple of 30 millivolts or less) and a voltage in the stopband of between 0 volts and 1 millivolts (i.e., a stopband ripple of 1 millivolts or less) is regarded as acceptable. Any voltage lower than 970 millivolts in the passband and any voltage above 1 millivolts in the stopband is regarded as unacceptable.

A fifth-order *elliptic (Cauer) filter* with a modular angle Θ of 30 degrees (i.e., the arcsin of the ratio of the boundaries of the passband and stopband) and a reflection coefficient p of 24.3 per cent is required to satisfy these design goals.

Since the high-level statement of behavior for the desired circuit is expressed in terms of frequencies, the voltage VOUT is measured in the frequency domain. SPICE performs an AC small signal analysis and report the circuit's behavior over five decades (between 1 Hz and 100 000 Hz) with each decade being divided into 20 parts (using a logarithmic scale), so that there are a total of 101 fitness cases.

Fitness is measured in terms of the sum over these cases of the absolute weighted deviation between the actual value of the voltage that is produced by the circuit at the probe point VOUT and the target value for voltage. The smaller the value of fitness, the better. A fitness of zero represents an (unattainable) ideal filter.

Specifically, the standardized fitness is

$$F(t) = \sum_{i=0}^{100} [W(d(f_i), f_i d(f_i)]$$

where f_i is the frequency of fitness case i; $d(x)$ is the absolute value of the difference between the target and observed values at frequency x; and $W(y,x)$ is the weighting for difference y at frequency x.

The fitness measure is designed to not penalize ideal values, to slightly penalize every acceptable deviation, and to heavily penalize every unacceptable deviation. Specifically, the procedure for each of the 61 points in the 3-decade interval between 1 Hz and 1000 Hz for the intended passband is as follows:

- If the voltage equals the ideal value of 1.0 volt in this interval, the deviation is 0.0.
- If the voltage is between 970 millivolts and 1 volt, the absolute value of the deviation from 1 volt is weighted by a factor of 1.0.
- If the voltage is less than 970 millivolts, the absolute value of the deviation from 1 volt is weighted by a factor of 10.0.

The acceptable and unacceptable deviations for each of the 35 points from 2000 Hz to 100 000 Hz in the intended stopband are similarly weighed (by 1.0 or 10.0) based on the amount of deviation from the ideal voltage of 0 volts and the acceptable deviation of 1 millivolts.

For each of the five 'don't care' points between 1000 and 2000 Hz, the deviation is deemed to be zero.

The number of 'hits' for this problem (and all other problems herein) is defined as the number of fitness cases for which the voltage is acceptable or ideal or that lie in the 'don't care' band (for a filter).

Many of the random initial circuits and many that are created by the crossover and mutation operations in subsequent generations cannot be simulated by SPICE. These circuits receive a high penalty value of fitness (10^8) and become the worst-of-generation programs for each generation.

For details, see Koza *et al.* 1996*b*.

5.4.2 Tri-state frequency discriminator

Fitness is the sum, over 101 fitness cases, of the absolute weighted deviation between the actual value of the voltage that is produced by the circuit and the target value.

The three points that are closest to the band located within 10 per cent of 256 Hz are 229.1 Hz, 251.2 Hz, and 275.4 Hz. The procedure for each of these three points is as follows: If the voltage equals the ideal value of 1/2 volts in this interval, the deviation is 0.0. If the voltage is more than 240 millivolts from 1/2 volts, the absolute value of the deviation from 1/2 volts is weighted by a factor of 20. If the voltage is more than 240 millivolts of 1/2 volts, the absolute value of the deviation from 1/2 volts is weighted by a factor of 200. This arrangement reflects the fact that the ideal output voltage for this range of frequencies is 1/2 volts, the fact that a 240 millivolts discrepancy is acceptable, and the fact that a larger discrepancy is not acceptable.

Similar weighting was used for the three points (2291 Hz, 2512 Hz, 2754 Hz) that are closest to the band located within 10 per cent of 2560 Hz.

The procedure for each of the remaining 95 points is as follows: if the voltage equals the ideal value of 0 volts, the deviation is 0.0. If the voltage is

within 240 millivolts of 0 volts, the absolute value of the deviation from 0 volts is weighted by a factor of 1.0. If the voltage is more than 240 millivolts from 0 volts, the absolute value of the deviation from 0 volts is weighted by a factor of 10. For details, see Koza *et al.* 1997*b*.

5.4.3 Computational circuit

SPICE is called to perform a DC sweep analysis at 21 equidistant voltages between –250 millivolts and +250 millivolts. Fitness is the sum, over these 21 fitness cases, of the absolute weighted deviation between the actual value of the voltage that is produced by the circuit and the target value for voltage. For details, see Koza *et al.* 1997*a*.

5.4.4 Robot controller circuit

The fitness of a robot controller was evaluated using 72 randomly chosen fitness cases each representing a different target point. Fitness is the sum, over the 72 fitness cases, of the travel times. If the robot came within a capture radius of 0.28 meters of its target point before the end of the 80 time steps allowed for a particular fitness case, the contribution to fitness for that fitness case was the actual time. However, if the robot failed to come within the capture radius during the 80 time steps, the contribution to fitness was 0.160 hours (i.e., double the worst possible time).

SPICE performs a nested DC sweep, which provides a way to simulate the DC behavior of a circuit with two inputs. It resembles a nested pair of FOR loops in a computer program in that both of the loops have a starting value for the voltage, an increment, and an ending value for the voltage. For each voltage value in the outer loop, the inner loop simulates the behavior of the circuit by stepping through its range of voltages. Specifically, the starting value for voltage is –4 volt, the step size is 0.2 volt, and the ending value is +4 volt. These values correspond to the dimensions of the robot's world of 64 square meters extending 4 meters in each of the four directions from the origin of a coordinate system (i.e., 1 volt equals 1 meter). For details, see Koza *et al.* 1997.

5.4.5 60 dB amplifier

SPICE was requested to perform a DC sweep analysis to determine the circuit's response for several different DC input voltages. An ideal inverting amplifier circuit would receive the DC input, invert it, and multiply it by the amplification factor. A circuit is flawed to the extent that it does not achieve the desired amplification, the output signal is not perfectly centered on 0 volts (i.e., it is biased), or the DC response is not linear. Fitness is calculated by summing an amplification penalty, a bias penalty, and two non-linearity penalties — each derived from these five DC outputs. For details, see Bennett *et al.* 1996.

5.5 Control parameter

The population size, M, was 640 000 for all problems. Other parameters were substantially the same for each of the five problems and can be found in the references cited above.

5.6 Implementation on parallel computer

Each problem was run on a medium-grained parallel Parsytec computer system (Andre and Koza 1996) consisting of 64 80-MHz PowerPC 601 processors arranged in an 8 by 8 toroidal mesh with a host PC Pentium type computer. The distributed genetic algorithm was used with a population size of Q = 10 000 at each of the D = 64 demes (semi-isolated subpopulations). On each generation, four boatloads of emigrants, each consisting of B = 2 per cent (the migration rate) of the node's subpopulation (selected on the basis of fitness) were dispatched to each of the four adjacent processing nodes.

6. Results

In all five problems, fitness was observed to improve over successive generations. A large majority of the randomly created initial circuits of generation 0 were not able to be simulated by SPICE; however, most were simulatable after only a few generations. Satisfactory results were generated in every case on the first or second trial. When two runs were required, the first produced an almost satisfactory result. This rate of success suggests that the capabilities of the approach and current computing system have not been fully exploited.

6.1 Lowpass filter

Many of the runs produced lowpass filters having a topology similar to that employed by human engineers. For example, in generation 32 of one run, a circuit (Fig. 8.8) was evolved with a near-zero fitness of 0.00781. The circuit was 100 per cent compliant with the design requirements in that it scored 101 hits (out of 101). After the evolutionary run, this circuit (and all evolved circuits herein) were simulated anew using the commercially available MicroSim circuit simulator to verify performance. This circuit had the recognizable ladder topology of a Butterworth or Chebychev filter (i.e., a composition of series inductors horizontally with capacitors as vertical shunts).

Figure 8.9 shows the behavior in the frequency domain of this evolved lowpass filter. As can be seen, the evolved circuit delivers about 1 volt for all frequencies up to 1000 Hz and about 0 volts for all frequencies above 2000 Hz.

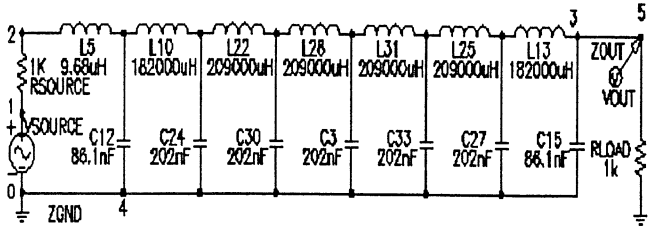

Fig. 8.8 Evolved 7-mng ladder lowpass filter.

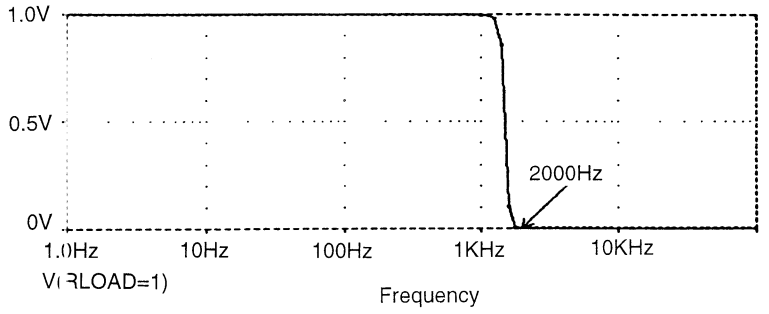

Fig. 8.9 Frequency domain behavior of genetically evolved 7-mng ladder lowpass f lter.

In another run, a 100 per cent compliant recognizable 'bridged T'; arrangement was evolved. In yet another run using automatically defined functions, a 100 per cent compliant circuit emerged with the recognizable elliptic topology that was invented and patented by Cauer. When invented, the Cauer filter was a significant advance (both theoretically and commercially) over the Butterworth and Chebychev filters.

Thus, genetic programming rediscovered the ladder topology of the Butterworth and Chebychev filters, the 'bridged T' topology, and the elliptic topology

6.2 Tri-state frequency discriminator

The evolved three-way tri-state frequency discriminator circuit from generation 106 scores 101 hits (out of 101). Figure 8.10 shows this circuit (after expansion of its automatically defined functions). The circuit produces the desired outputs of 1 volt and 1/2 volts (each within the allowable tolerance) for the two specified bands of frequencies and the desired near-zero signal for all other frequencies.

6.3 Computational circuit

The genetically evolved computational circuit for the square root from generation 60 (Fig. 8.11), achieves a fitness of 1.68, and has 36 transistors, two

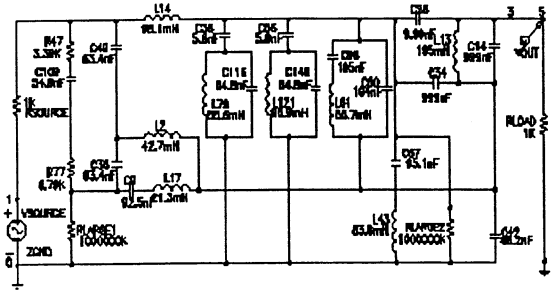

Fig. 8.10 Evolved frequency discriminator.

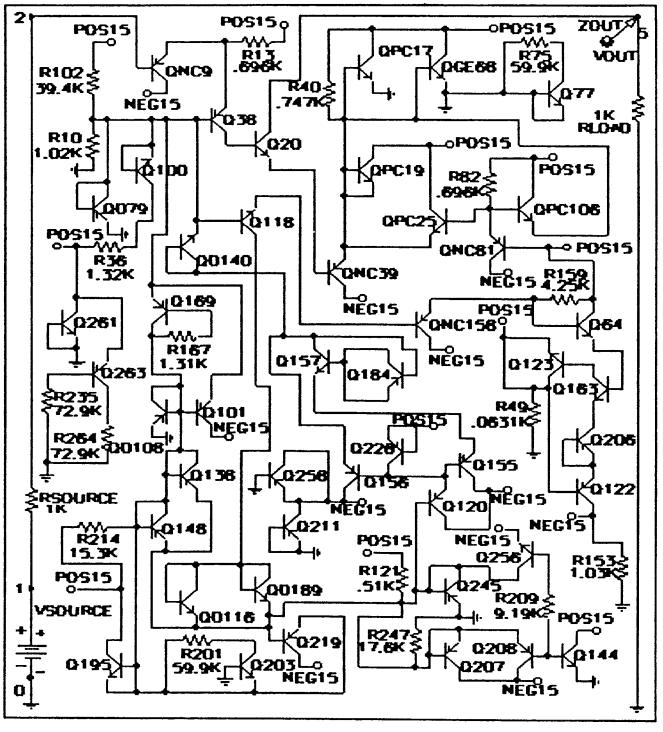

Fig. 8.11 Evolved square root circuit.

diodes, no capacitors, and 12 resistors (in addition to the source and load resistors in the embryo). The output voltages produced by this best-of-run circuit are almost exactly the required values.

6.4 Robot controller circuit

The best-of-run time-optimal robot controller circuit (Fig. 8.12) appeared in generation 31, scores 72 hits, and achieves a near-optimal fitness of

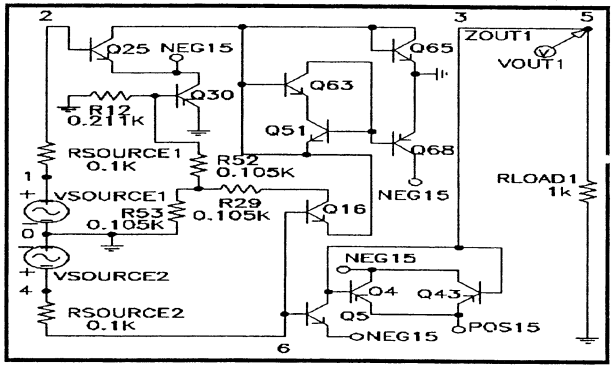

Fig. 8.12 Evolved robot controller.

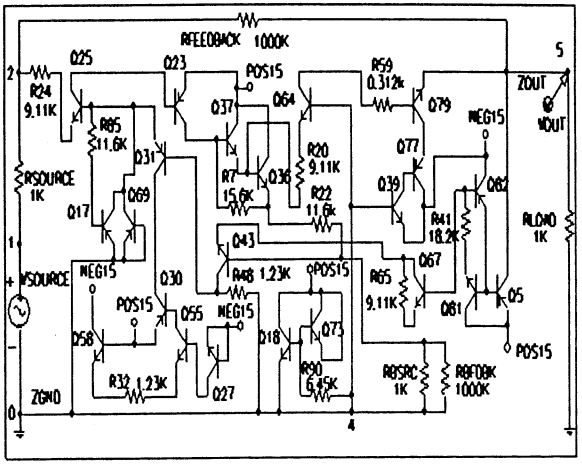

Fig. 8.13 Genetically evolved amplifier.

1.541 hours. In comparison, the optimal value of fitness for this problem is known to be 1.518 hours. This best-of-run circuit has 10 transistors and 4 resistors. The program has one automatically defined function that is called twice (incorporated into the figure).

The best circuit from generation 109 (Fig. 8.13) achieves a fitness of 0.178. Based on a DC sweep, the amplification is 60 dB here (i.e., 1000-to-1 ratio) and the bias is 0.2 volt. Based on a transient analysis at 1000 Hz, the amplification is 59.7 dB; the bias is 0.18 volts; and the distortion is very low (0.17 per cent). Based on an AC sweep, the amplification at 1000 Hz is 59.7 dB; the flatband gain is 60 dB; and the 3dB bandwidth is 79 333 Hz. Thus, a high-gain amplifier with low distortion and acceptable bias has been evolved.

7. Other circuits

Numerous other circuits have been similarly designed, including asymmetric bandpass filters (Koza *et al.* 1996*c*), crossover filters (Koza *et al.* 1996*a*), double passband filters (Koza *et al.* 1996), amplifiers (Koza *et al.* 1997), a temperature-sensing circuit, and a voltage reference circuit (Koza *et al.* 1997).

8. Conclusion

Genetic programming evolved the topology and sizing of five different prototypical analog electrical circuits, including a low pass filter, a tri-state frequency discriminator circuit, a 60 dB amplifier, a computational circuit for the square root, and a time-optimal robot controller circuit. The problem-specific information required for each of the eight problems is minimal and consists primarily of the number of inputs and outputs of the desired circuit,

the types of available components, and a fitness measure that restates the high-level statement of the circuit's desired behavior as a measurable mathematical quantity. All five of these genetically evolved circuits constitute instances of an evolutionary computation technique solving a problem that is usually thought to require human intelligence.

References

Aaserud, O. and Nielsen, I. Ring. (1995). Trends in current analog design: A panel debate. *Analog Integrated Circuits and Signal Processing*, **7**(1) 5–9.

Andre, David and Koza, John R. (1996). Parallel genetic programming: A scalable implementation using the transputer architecture. In Angeline, P.J. and Kinnear, K.E. Jr. (eds). *Advances in genetic programming 2*. MIT Press, Cambridge.

Angeline, Peter J. and Kinnear, Kenneth E. Jr. (eds). (1996) *Advances in genetic programming 2*. MIT Press, Cambridge, MA.

Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. (1997). *Genetic programming — an introduction*. Morgan Kaufmann and Heidelberg, San Francisco, CA.

Bauer, R.J., Jr. (1994). *Genetic algorithms and investment strategies*. John Wiley.

Bennett III, Forrest H, Koza, John R., Andre, David, and Keane, Martin A. (1996). Evolution of a 60 Decibel op amp using genetic programming. In Higuchi, Tetsuya, Iwata, Masaya, and Lui, Weixin (eds). *Proceedings of international conference on evolvable systems: from biology to hardware (ICES-96)*, Lecture Notes in computer Science, Volume 1259. Springer-Verlag, Berlin. pp. 445–69.

Bhanu, Bir and Lee, Sungkee. (1994). *Genetic learning for adaptive image segmentation*. Kluwer Academic Publishers, Boston.

Brave, Scott. (1996). Evolving deterministic finite automata using cellular encoding. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (eds), pp. 39–44. *Genetic programming 1996: proceedings of the first annual conference, July 28–31, 1996, Stanford University*. MIT Press, Cambridge, MA.

Davidor, Yuval. (1991). *Genetic algorithms and robotics*. World Scientific, Singapore.

Gen, Mitsuo and Cheng, Runwei. (1997). Genetic algorithms and engineering design. John Wiley and Sons, New York.

Goldberg, David E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA.

Grimbleby, J.B. (1995). Automatic analogue network synthesis using genetic algorithms. *Proceedings of the first international conference on genetic algorithms in engineering systems: innovations and applications*, Institution of Electrical Engineers, London.

Gruau, Frederic. (1992). *Cellular encoding of genetic neural networks*. Technical report 92–21. Laboratoire de l'Informatique du Parallélisme. Ecole Normale Supérieure de Lyon. May 1992.

Holland, John H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI.

Kinnear, Kenneth E. Jr. (editor). (1994). *Advances in genetic programming*. The MIT Press, Cambridge, MA.

Kitano, Hiroaki, (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4(1990) 461–76.

Koza, John R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA.

Koza, John R. (1994*a*). *Genetic programming II: automatic discovery of reusable programs*. MIT Press, Cambridge, MA.

Koza, John R. (1994*b*). *Genetic programming II videotape: the next generation*. MIT Press, Cambridge, MA.

Koza, John R. (1995). Evolving the architecture of a multi-part program in genetic programming using architecture-altering operations. In McDonnell, John R., Reynolds, Robert G., and Fogel, David B (eds). *Evolutionary programming IV: proceedings of the fourth annual conference on evolutionary programming*, pp. 695–717. The MIT Press, Cambridge, MA.

Koza, John R., Andre, David, Bennett III, Forrest H., and Keane, Martin A. (1996). Use of automatically defined functions and architecture-altering operations in automated circuit synthesis using genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (eds). *Genetic programming 1996: proceedings of the first annual conference*. The MIT Press, Cambridge, MA.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. (1996a). Four problems for which a computer program evolved by genetic programming is competitive with human performance. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pp. 1–10. IEEE Press.

Koza, John R., Bennett III, Forrest H., Andre, David, and Keane, Martin A. (1996b). Automated design of both the topology and sizing of analog electrical circuits using genetic programming. In Gero, John S. and Sudweeks, Fay (eds). *Artificial intelligence in design '96*, pp. 151–70. Kluwer, Dordrecht.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. (1996c). Automated WYWIWYG design of both the topology and component values of analog electrical circuits using genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (eds). *Genetic programming 1996: proceedings of the first annual conference*. The MIT Press, Cambridge, MA.

Koza, John R., Bennett III, Forrest H., Andre, David, and Keane, Martin A. (1997). Evolution using genetic programming of a low-distortion 96 Decibel operational amplifier. *Proceedings of the 1997 ACM symposium on applied computing, San Jose, California, February 28–March 2, 1997*, pp. 207–16. Association for Computing Machinery.

Koza, John R., Bennett III, Forrest H., Andre, David, Keane, Martin A., and Dunlap, Frank. (1997). Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*, 1(2), 109–28.

Koza, John R., Bennett III, Forrest H., Keane, Martin A., and Andre, David. (1997). Automatic programming of a time-optimal robot controller and an analog electrical circuit to implement the robot controller by means of genetic programming. *Proceedings of 1997 IEEE international symposium on computational intelligence in robotics and automation*, pp. 340–6. Computer Society Press, Alamitos, CA.

Koza, John R., Bennett III, Forrest H., Lohn, Jason, Dunlap, Frank, Andre, David, and Keane, Martin A. (1997). Automated synthesis of computational circuits using genetic programming. *Proceedings of the 1997 IEEE conference on evolutionary computation*, pp. 447–52. IEEE Press, Piscataway, NJ.

Koza, John R., Bennett III, Forrest H., Lohn, Jason, Dunlap, Frank, Andre, David, and Keane, Martin A. (1997b). Use of architecture-altering operations to dynamically adapt a three-way analog source identification circuit to accommodate a new source. In Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max,

Iba, Hitoshi, and Riolo, Rick L. (eds). *Genetic programming 1997: proceedings of the second annual conference, pp. 213-21. Morgan Kaufmann, San Francisco, CA.*

Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L. (eds). *Genetic programming 1997: proceedings of the second annual conference*. Morgan Kaufmann, San Francisco, CA.

Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (eds). (1996). *Genetic programming 1996: proceedings of the first annual conference*. The MIT Press, Cambridge, MA.

Koza, John R., and Rice, James P. (1992). *Genetic programming: the movie*. MIT Press, Cambridge, MA.

Mruiskamp Marinum Wilhelmus and Leenaerts, Domine. (1995). DARWIN: CMOS opamp synthesis by means of a genetic algorithm. *Proceedings of the 32nd design automation conference*, pp. 433–8. Association for Computing Machinery, New York, NY.

Man, K.F., Tang, K.S., Kwong, S., and Halang, W.A. (1997). *Genetic algorithms for control and signal processing*. Springer-Verlag, London.

Michalewicz, Z. (1992). *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, Berlin.

Mitchell, Melanie. (1996) *An introduction to genetic algorithms*. The MIT Press, Cambridge, MA.

Ohno, Susumu. (1970). *Evolution by gene duplication*. Springer-Verlag, New York.

Quarles, Thomas, Newton, A.R., Pederson, D.O., and Sangiovanni-Vincentelli, A. (1994). *SPICE 3 Version 3F5 User's Manual*. Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA. March 1994.

Rutenbar, R.A. (1993). Analog design automation: Where are we? Where are we going? *Proceedings of the 15th IEEE CICC*. IEEE, New York. 13.1.1–13.1.8.

Samuel, Arthur L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3). 210–29.

Stender, Joachim (editor). (1993). *Parallel genetic algorithms*. IOS Publishing, Amsterdam.

Stnder, Joachim, Hillebrand, and Kingdon, J. (editors). (1994). *Genetic algorithms in optimization, simulation, and modeling*. IOS Publishing, Amsterdam.

Thompson, Adrian. (1996). Silicon evolution. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (eds). *Genetic programming 1996: proceedings of the first annual conference*. MIT Press, Cambridge, MA.

Index