About this book

An Expert System is, simply, computer software that encapsulates some area of human expertise. It can help a doctor to diagnose illnesses, a chemist to deduce chemical structures or help to choose the best place to drill for oil or precious metals. Up to now, this has been the province of big computers, bigger programs, and exotic languages. But, this book brings the important topic down to earth with easy explanations and practical BASIC programs that you can key in immediately on your personal microcomputer.

You too can join in the debate about Artificial Intelligence. You can see for yourself how an Expert System works, and you can use it on your own microcomputer to help you in such tasks as weather forecasting, car maintenance, even medical diagnosis. All of the programs in the book have been tested on the popular Apple II and Sinclair Spectrum micros.

About the author

Chris Naylor holds degrees in Psychology and Philosophy from the University of Keele and Mathematics and Statistics from the University of London.

He is a member of The British Psychological Society, The Institute of Mathematics and its Applications, The Institute of Statisticians, The Institute of Personnel Management, The Institute of Data Processing Management and the British Computer Society.

He has been using computers since 1965 and, for the last ten years, has written on various computing topics in magazines including: Computer Digest, Which Computer, Computer Management, Datalink, Computer Talk, Computerworld, and Practical Computing.

"Can we have more along the lines of ***Build Your Own Expert System*** *by Chris Naylor?"*
Computing Today.

"Naylor's book serves as a good introduction to the subject and includes descriptions of most of the well-known expert systems in operation currently."
The Times.

"This is an excellent book on a forbidding subject. Even if you don't build an expert system from it, your money won't have been wasted."
Personal Computer News.

"This is probably the most unusual text book you will ever read. If you're jaded and looking for something new to try, you couldn't do better."
Which Micro & Software Review.

"Build Your Own Expert System by Chris Naylor is one of the most interesting new books I've read recently."
Popular Computing Weekly.

Published in the UK by **SIGMA PRESS**, Wilmslow

Published in the USA and Canada by **HALSTED PRESS**, a division of
JOHN WILEY & SONS,
New York · Chichester · Brisbane · Toronto

Sigma Press Edition ISBN 0-905104-41-2
Halsted Press Edition ISBN 0-470-20172-X

Build your own Expert System

Chris Naylor



rtificial ntelligence or the aspiring icrocomputer

With listings for Apple II and Sinclair Spectrum

SIGMA PRESS

HALSTED PRESS

Build Your Own Expert System

by

Chris Naylor

SIGMA PRESS Publishers. Wilmslow

HALSTED PRESS a division of JOHN WILEY & SONS
New York · Chichester · Brisbane · Toronto

IN 8519

Copyright © 1983 by C. M. Naylor
Reprinted 1984, 1985.

All Rights Reserved

No part of this book may be reproduced or transmitted by any means without the prior permission of the publisher. The only exceptions are for the purposes of review, or as provided for by the Copyright (Photocopying) Act or in order to enter the programs herein onto a computer for the sole use of the purchaser of this book.

ISBN 0-905104-41-2 (Sigma Press)
ISBN 0-470-20172-X (Halsted Press)

Kyb
470
92

First published in 1983 by:
SIGMA PRESS,
5 Alton Road,
Wilmslow,
Cheshire,
U.K.

Universitäts- und Landesbibliothek Sachsen-Anhalt Bibliothek

Published in 1985 in the USA and Canada by:
HALSTED PRESS,
a Division of **JOHN WILEY & SONS, INC**
New York.

Distributed in Europe and Africa by:
JOHN WILEY & SONS LIMITED,
Baffins Lane, Chichester,
West Sussex, England.

Printed in Great Britain by J. W. Arrowsmith Ltd., Bristol BS3 2NT

CONTENTS

1. Why 'Expert Systems'? 1

1.1 What do You Want an Expert System for? 3
1.2 What do Other People Want an Expert System for? 4
1.3 What is an Expert System? 5
1.4 What do You Want Your Expert System to do? 9
1.5 Some Untrue Things about Expert Systems 11

2. A Statistical Scheme 13

2.1 Setting up a Matrix 13
2.2 Probabilities 20
2.3 More Probabilities 23
2.4 More Variables 26
2.5 Bayes' Theorem 33

3. Avoiding Probabilities 35

3.1 How to Make the Computer do the Hard Work 36
3.2 The Learning System 37
3.3 Other Types of Data 43
3.4 The Judgement Rule 45
3.5 Building a Rule 49
3.6 Prior Probabilities 53
3.7 Expanding your Options 54
3.8 Can it Make a Mistake? 58
3.9 Summing Up: The Program so far 61

4. Improving your Expert 65

4.1 Parallel and Sequential Decisions 65
4.2 Adding some Commonsense 72
4.3 A Trial Run of our New Expert 78

5. The Making of A Real-World Expert 89

5.1 The Weather Again 89
5.2 A Chi-Squared Program 98
5.3 Exercising your Expert 99
5.4 Direct Estimation 105

6. Running for Real 110

6.1 Using your Expert 110
6.2 Reserved Judgement 115
6.3 The Problem of Distance 117
6.4 Understanding your Problem 124

7. An Expert on Everything in the Entire Known World 125

7.1 Nodes 125
7.2 The Variables so Far 131
7.3 Going through the Nodes 137
7.4 Tailor-made Nodes 142
7.5 Specific Code 145
7.6 Saving your Expert 146
7.7 The Multi-Node Code 148
7.8 Some Examples 157

8. How can you Use your Expert 164

8.1 Choosing a Problem 164
8.2 Analysing the Problem 166

9. Large - Scale Expert Systems 170

9.1 MYCIN - Medical Diagnoses 170
9.2 PUFF - Breathing Disorders 178
9.3 DENDRAL - Chemical Structures 185
9.4 PROSPECTOR - Searching for minerals 190
9.5 Some Other Examples 196

10. A Rule-Based BASIC Expert 201

10.1 A System that Works Backwards 201
10.2 The BASIC Program 205
10.3 A Medical Knowledge Base 214

11. The Tower of Babel 221

12. Summary and Technical Overview 229

12.1 Events 229
12.2 Probabilities 229

12.2.1 Bayes' Theorem 230
12.2.2 Prior and Posterior Probabilities 231
12.2.3 Odds 232
12.2.4 Approximations 232
12.2.5 Combinations 233
12.2.6 Descriptive Statistics 233
12.2.7 Normal Distribution 234
12.2.8 Discrete and Continuous Variables 235

12.3 Surfaces 235
12.4 Discrimination 236
12.5 The Learning Algorithm 237
12.6 Parallel and Sequential Procedures 237
12.7 Maximum and Minimum Values 238
12.8 Processing Strategies 238

12.8.1 Goal Driven Strategies 238
12.8.2 Data Driven Strategies 239
12.8.3 Selecting the Next Variable 239

12.9 Intermediate Conclusions 240

12.9.1 Explanatory Systems 241

12.10 Linear Interpolation of Responses 242
12.11 Data Formats

13. Select Readings 244

Index 247

For Catherine and Philip

and much good may it do them...

PREFACE

A potential reader is standing in a bookshop with a copy of a possible purchase in his or her hand. Having looked at the front cover, and looked at the back cover, he or she now reads the Preface to determine whether or not the book should be purchased. The Preface, says the Theory, clinches the sale.

So much for theory. The real reason you should buy this book is because, for a book on computers, it is relatively cheap. It also contains working examples in BASIC for both the Apple II and the Sinclair Spectrum so you get some 'free' programs thrown in for your money.

It tells you a moderate amount about Expert Systems, but, frankly, to disclose exactly what it does tell about Expert Systems would rather annihilate the reason for buying it. After all, you could just stand there, in the bookshop, reading the preface and hang onto your money. But it will (just as a sort of an appetiser) enable you to build your own medical diagnosis program. Or a program for working out why your car won't start in the morning. Or it will enable you to build a learn-by-example Expert System which can be taught expertise in a wide range of areas.

It will also teach you a fair amount about statistics and inferencing systems but, despite that, you should still shell out the necessary and buy a copy.

I know money is tight these days and you could usefully spend it on something else, like drink, which would give *you* greater pleasure but I have to make a living too you know and the cost of typewriter ribbons alone was pretty enormous when it came to bashing this lot out.

Well, after all that, maybe you've had the heart to fork out on a copy and you're actually planning to read the thing now. What you do is start at the beginning and carry on until you get to the Technical Overview. Then think of something you'd like to try out — such as a learning system or a bit of problem diagnosis — and dig out the relevant parts using the contents pages, the index, and the technical overview to tie it together.

Alternatively, you could have your computer switched on as you read the book and key in the examples as you go along, that way seeing how they work as an aid to understanding. If you do this, it could take you ages to get to the end, of course.

The big point to note though is that this book is not arranged like a normal text book. The chapters are not isolated entities. One way or another you do have to churn right through from front to back to get the ideas in the proper sequence. A few people have commented that it reads more like a novel than a text book in the way it's arranged - which is a fair point, but it's probably as well to warn you that it's like that.

Prior to publication this book was read by Graham Beech, Phil Bradley, Bill Hudspith and Phil Manchester (in alphabetical order).

They all chipped in with comments of one sort or another and they each have caused some improvements to be made to the final version. Which was nice of them.

If anyone, having read the book, has any comments which might lead to useful future alterations, then drop the publisher a line and let him know.

Chris Naylor, 1983

CHAPTER 1

Why 'Expert Systems'?

"An expert system is regarded as the embodiment within a computer of a knowledge-based component from an expert skill in such a form that the system can offer INTELLIGENT ADVICE or take an INTELLIGENT DECISION about a processing function. A desirable additional characteristic, which many would consider fundamental, is the capability of the system, on demand, to JUSTIFY ITS OWN LINE OF REASONING in a manner directly intelligible to the enquirer. The style adopted to attain these characteristics is RULE-BASED PROGRAMMING".

A formal definition of expert systems approved by the British Computer Society's committee of the specialist group on expert systems.

Once upon a time, a long time ago when the Earth was still new and the Sun had a big smile on it's face when it got up each morning, there was no such thing as Expert Systems. Yet, suddenly, everyone seems now to be talking about the things. Why is this?

Well, the answer is that Scientists Have Determined that there is an increasing quantity of Government Money around for computer applications and, further, they have also determined that there is scant chance of accessing this pile of money unless there is a goodish chunk of expensive and arcane research activity to be carried out on the application of computers. So, to this end, they invented Expert Systems which, because nobody in Governmental circles knows what they are, are liable to attract Government Money if only as an aid to identification.

Less mercenary scientists (whose names currently escape one) are not interested in Government Money *per se* and will, in fact, point to the notable absence of Government Money to date in this field. These altruists will simply state that they want to make computers more accessible to more people. They want to make computers *think* like people. They want computers to *replace* people. They want computers to be *user-friendly.* And, in the final analysis, they probably also want computers to take on the job of allocating Government Money.

All of this is fine - but the real problem lies in finding out enough about Expert Systems so that one can even begin to attract Government Money on one's own behalf. Beyond this point in the book the subject of Money will receive little attention but, hopefully, the book will enable you to get the hang of Expert Systems so that you can, at least, build your own - Government Funded or not.

The first part of the book relies on your total ignorance to build an Expert System - it is a learn-by-example system which can pick up skills in a wide range of subject areas. It *acquires* expertise.

Later, some knowledge of a specific area is assumed and we build an Expert System which is able to use this knowledge intelligently to offer *advice* after the fashion of a human expert.

A point which really must be made is this: that this book, as an aid to understanding and as a sop to sloth, gives all of its examples in BASIC. But it does not contain an overview of, or instructions on, how to program in BASIC. That is the one piece of expertise which the reader has to bring to the task for him or herself.

The main examples are given in BASIC with the major programs written in dialects for both the Apple II and the Sinclair Spectrum so they should be readily modifiable onto most micros. Very short sections of code are written just in Apple BASIC (i.e. Applesoft.)

The final chapter in the book summarises the information given in previous pages so that, when you get that far, you'll be able to see exactly what it was that you were reading about earlier.

1.1 What do you want an expert system for?

There are two major faults possessed by most existing expert systems and these two faults are: that you, personally, don't understand how they work, and that you, personally, haven't got one.

These faults can, in extreme cases, be quite serious.
You slink around avoiding other people's eyes. You avoid conversation with others. You hide behind COBOL manuals. You listen greedily to other's talk of 'knowledge bases', 'artificial intelligence' and 'real-world representations'. You are afraid to come clean and ask what it's all about for fear of social rebuff.

You become an outcast and a despised person in your own eyes.

All of which can become a bit irksome after the first five minutes or so. Particularly when you feel that the subject matter can't really be all that difficult as evidenced by the fact that those who do profess to understand it can't, surely, be any cleverer than you yourself are.

What is needed, to put matters right, is for you to have your Very Own Expert System. Tailor-made to A Little Known Design it will enable you to lean confidently on any bar counter and pontificate on the subject of expert systems. Instead of hiding your head and keeping your own counsel you will be able to raise a slight sneer in the direction of those who have expert systems but didn't build them themselves. You will be able to laugh condescendingly at those who don't really understand how their own expert system works. And at those who actually don't have an expert system of any kind (one doesn't even begin to know how such people manage to get by) you will be able to raise a deeply quizzical eyebrow.

For you (the tall, confident one standing near the centre of the bar and holding forth to a crowd of admirers, many of whom are young and nubile) are about to Build Your Own Expert System.

No special knowledge is required although it would be handy if you had access to a computer; otherwise much of what is to follow may come to seem, somehow, well, Academic.

1.2 What do other people want an expert system for?

There are two main uses for expert systems and these correspond to the two concepts of manifest and latent functions in sociology.

The manifest function of an expert system is to provide, on a computer, human expertise. For instance, they can diagnose illness, deduce chemical structures, suggest sites for digging up precious metals or carry out a host of similiar tasks. They are user-friendly to some degree - embodying human knowledge in a form vaguely similiar to the form in which a human expert might hold knowledge. They often have some ability to explain their actions and opinions in much the same way that a human expert might. And, like a human expert, they might even be able to teach their expertise to someone.

The other function of expert systems - hinted at earlier - is their latent function which is, one suspects, to baffle the ignorant with arcane explanations of how they were built. Typically, they use large computers, of the sort you have not got, and employ exotic-ish languages, such as LISP and PROLOG, which you have not got either. This tends to have the advantage of somewhat sewing up the market for supplying expert systems because, if demand can be stimulated by a description of an expert system's manifest functions, this demand - surely? - can't be met simply by anyone possessed of a micro and a BASIC interpreter. Which doubtless affects the price.

The way out of this problem is to provide a micro-orientated guide to building expert systems. Not a complete guide which gives the last word on the subject necessarily - but enough to break down some of the mystique and get the average person started.

Up to now the real problem in understanding expert systems has been simply that there was no simple place to go for an introduction to the subject. Unless you were prepared to put some pretty hard thought into the matter, all of the currently available writings on the subject just seemed to emphasise the difficulty of ever understanding any aspect of the subject - which tends to inspire one to give up and leave it to the experts! It's all rather reminiscent of trying to climb a mountain (well, hill, maybe) on which there is just no first toehold to be found. If you could only get started you'd feel a lot more optimistic about what might be done and you'd even be able to work out a lot of the subsequent steps for yourself without the aid of a book. And, after all, that's what the human experts on expert systems are doing: working it out for themselves.

The trick lies in getting yourself into the same way of thinking as the current leaders in the field. To realise that there is something concrete and accessible to think about with real toeholds to hang onto. To think that there are some interesting problems here and that these problems, with a bit of thought, are perfectly soluble and that the solutions as they appear are perfectly amenable to being explained in plain language. That the subject of expert systems is not in fact a mystic art but, like all computer subjects, is something as practical and down-to-earth as carpentry.

1.3 What is an expert system?

Before you climb into your overalls and rush to your workbench it's as well to just pause for a moment and ask: what, actually, is an expert system? For, if you think about it, you'll realise that the answer to this question might well have a profound effect on the finished object. Actually, the answer won't affect the object much - as we shall see - but, certainly, you'd think it should.

It all started many years ago, back in the empty vastnesses of time when computers were about as powerful as a pocket calculator and transistors were spoken of in awed tones. In between being struck speechless by the power and complexity of the monsters with which they worked, computer scientists found words to express their deepest desires.

"Wouldn't it be nice," they used to say to each other, "if we could get this thing to do something other than payroll calculations."

"Yes," they used to agree, "we've got literally hundreds of words of memory and it now works faster than the chief accountant yet, for all that, it is like working with an idiot."

"Certainly he is an idiot," another would concede, "but our computer is little better than him."

And they would drink beer long into the night and attempt to hatch schemes whereby the full power of their Frankensteinian monster could be unleashed.

Well, they are still sitting there, drinking beer and plotting against the chief accountant and the computer is still doing the payroll. The progress has not been dramatic in the direction of getting the computer to do something more, but the motivation is still the same. For, like Frankenstein, they wanted to dream up a way of breathing a little life into

their subject. They didn't just want a glorified adding machine, they wanted an actual thinking machine.

That, of course, is how the whole field of artificial intelligence started up, and the field of expert systems was just a part of it.

People noticed that the chief accountant, say, wasn't just a glorified adding machine (despite the rumours) but that he was something of an expert in his own field. He could cook the books, for instance, in a way quite beyond the abilities of any computer.

And it wasn't just accountants that could beat computers. In every field there were human experts who had special skills and knowledge which made them both indispensable and expensive. Every time a group of computer scientists talked to someone who was an expert in some field they would listen to him for, maybe, a couple of hours. Then they would start to long for the day when he could be replaced by a computer so that they could switch him off, and then forget to pay him.

The dream was an alluring one. So alluring that, in time, it took sufficient hold for people to think that there must be some way of realising that dream. The problem was: how?
It's easy to see that, if one did solve the problem, one could call the result an 'expert system'. It would have the expertise previously found in human experts yet it could be switched off at night. But, the problem as to how this was to be done not only proved hard to solve - it has hardly been solved yet.

Ask anyone how a human expert works and, apart from comments like 'slowly', you'll find that nobody knows. Or, at least, they don't know to the extent that you could write a computer program to do it. Any explanation will be fuddled with words like 'judgement' and 'experience' which simple don't occur in the world of formal languages.

But the basic ideas are simple enough.

People, it's said, are general purpose thinking machines. Give them any problem and a bit of experience and they can use their judgement to think out a satisfactory solution to the problem. All that people consist of is a collection of brain cells, wired together somehow, and all that computers consist of is a collection of memory cells, also wired together somehow. So, write a program which will solve problems (in general) and you have a system which will replace people. With the added advantages of not breaking down, forgetting, going wrong, wanting to be paid, and so on. The problem is that this general purpose problem solver has been pretty elusive.

Less elusive have been specific problem solvers. For example, you might have a calculation to perform. Well, you could easily write a program to do that for you. In fact, if you've used a computer you already have done that. Give your machine an arithmetic expression and it can work out the answer for you. That sounds pretty obvious but in evaluating an arithmetic expression the computer has done everything which a human mathematician would have done. It hasn't just added a few numbers together, it's taken a string of symbols and manipulated them to put them into a sensible form. After that, any arithmetic would have been really trivial.
And if you should think that this is beside the point, consider a problem which your current computer can't immediately handle. Take the problem of integrating a mathematical expression. Integrating expressions can be notoriously difficult to do with some functions - so you might reasonably think that it would be a suitable topic for an expert system. But now suppose that you had written such a system. What would it actually look like?

It would look, probably, just like your current computer with its BASIC interpreter.The only difference would be that it had a few more expressions in the language. In fact, if you look around at some pocket calculators, you'll find that they've got 'integrate' keys which will carry out numerical integration for you on any function. So there's really nothing remarkable about that anymore.

And this serves to illustrate a point: once we know how to do something and can write a program for it, then it all ceases to appear at all remarkable. Only a short while ago, if you didn't happen to be a maths graduate yourself, you'd have had to call one in to perform integration. In other words, you'd have sent for an expert. With the calculators and programs available today that isn't necessary because the task has ceased (virtually) to be one which needs an expert. And, as a consequence, we'd hardly call the 'integrate' key on a pocket calculator an expert system.

To take a more everyday example, there once were people who were payroll experts. In fact, some of the most agile minds in the British Empire could have been found in the Army Pay Corps. Adept at finding loopholes, special cases, hitherto unknown allowances, these human experts (some would say 'superhuman') held sway for years. It was simply the payroll program which sent pay clerks the way of the dinosaur. The payroll program can easily be seen as an example of an expert system - it embodies the total sum of human expertise on the subject of payrolls. But nobody thinks of it like that anymore.

It's simply another case of a problem seeming to be trivial once someone has solved it.

And that's really the case with expert systems today. Previous advances are dismissed as scarcely being advances at all for the very simple reason that, like the early computer scientists, we're all sitting around wishing that we could get the computer to do something 'special'. Something more than it's doing already. But, if it did, we'd still be sitting around making noises of discontent because we'd still be looking for that something extra which, as yet, we have not got.

Each advance brings with it another computer program - no more and no less than that. But there just doesn't seem to be that one, final, Big Advance which brings the ultimate program. The one that finally breathes life into the monster.

And, just to make matters worse, if there were such a system, the next thing that would happen is that there would appear a new breed of experts. Human experts, they would be expert in the workings of the ultimate systems. They would know more about it than did anyone else - and they would be well-paid for their expertise. And, if you don't immediately believe that, ask yourself how it was that computer programmers ever appeared on the scene.

Suppose, for instance, that you devise an expert system which is able to carry out medical diagnosis. This has, of course, been done already. What do you think would happen next? Well, learned papers would appear in academic journals describing the system and saying what it could do. And others, noting what it couldn't do would set about building a better expert system. This, of course, has already happened. It is past history. You might sum it up by saying that: a computer program was written which was then criticised and improved by human experts who then wrote a better computer program. And so on. ...

Inevitably then, any expert system is only going to be a temporary palliative. Something to stave off the pangs of deprivation for a while. The trick is to find a good palliative rather than a bad one. Something that makes you think 'That's clever!' - even if you don't always continue to think it's clever. A computer program, in short, which will do something that you hadn't realised could be done by a computer at all. A computer program which does something which you'd have thought really needed the services of a human expert to achieve.

But still, in the end, a computer program.

1.4 What do you want your expert system to do?

Having defined an expert system as a computer program, no more, no less, we could just sit around content in the knowledge that we knew what an expert system was and we already had one. This is the easy way out.

The hard way out is to take the functional approach and ask: what do you want your expert system to do?

This is a dangerous question to ask because the answers can involve you in having to do some work, something which is generally considered distasteful. But that's computers for you. Generally speaking, people don't define computer programs except by virtue of what they do. So, if expert systems are computer programs then, what do they do?

At this point the onus goes over, temporarily, to you. Because the question is not "What does one, in general, want an expert system to do?" It's "What do you, in particular *you,* want your very own expert system to do?" After all, it's you who's going to build it. You might as well have some say in the matter if you can.

Well that question is, of course, easy. Settling into your armchair you switch on your computer, close your eyes and dream. The room swims before your eyes, a warm glow passes through your very being. A relaxed, confident smile plays on your lips as you idly key in the question: HOW CAN I BECOME A MILLIONAIRE? and a few simple, well-chosen phrases appear on the screen by way of answer.

"Of course!" you exclaim. "Yes, certainly. Yes. That would obviously make me a millionaire. If only I'd built my own expert system sooner."

And you make a note of the answer and proceed to interrogate that machine on the previously-vexed question of how you can stop your hair going grey, followed by a session on how to stop dandelions growing in your lawn.

It all works like a dream and, in fact, it is a dream.

Which is a bit of a pity, really. But Life's like that and no amount of programming is really going to help that much.

The problem is that some things are impossible to program, expert system or no expert system, and if you want to program an impossible thing then you will encounter difficulties of implementation.

Returning from dreams of great things to the real world, in which you're

sitting there with a computer in front of you, needn't be too much of a let-down.

Having said that some things are impossible to program doesn't mean that everything's impossible to program. But why, you might ask, talk about programming when what you really want is to get on with building your very own expert system? Well, it's really just a matter of making sure that 'expert systems' doesn't get mistaken for the phrase 'universal panacea'. Take an analogous case which has been around a bit longer: databases.

Databases can be pretty complex things. They can take a lot of working out and a lot of understanding. This can lead to paralysis of the intellectual kind on the part of those who encounter databases for the first time. Yet, if you have a few files (and who doesn't) then you have a database (of sorts). So where's the problem? To be honest, there isn't a problem. You just ease into the subject kind of gradual and you soon get the hang of it.

It's much the same with expert systems. There's nothing much to them, really. Which is a good thing, seeing as how you're planning to build one.

There is one idea which is worth having in mind though: that's the idea of an expert system as being a system which has 'judgement'. Now, all computer programs tend to have some measure of judgement. As soon as you write code to compare one value with another and take specific action depending on the outcome you have judgement.

It's just that expert systems tend to place judgement rather centrally in their design. Judgement, rather than calculation, tends to typify expert systems. But, as the judgement usually comes as a result of calculation, the difference tends to be conceptual rather than actually, as it were, real.

So, think of something you'd like an expert system to do and ask yourself if it can be reduced to a series of judgements. If it can, you have a good chance of building an expert system to do it. And that allows an awfully wide range of possibilities.

To start you off, here are some possibilities:

Diagnosis of common illnesses.
Fault finding in simple circuits, or even a TV.
Diagnosing plant diseases.
Electrocardiogram recordings.
Classifying animals, birds or plants according to species.

But, you are certain to have your own ideas.

1.5 Expert systems: some untrue things

On the subject of expert systems, people say a wide range of things and not all of them are factually true.

The common error, strangely enough, is not to overstate what an expert system could, in principle, do. It is to understate the possibilities.

Typically, you might hear that "an expert system can only do ..." followed by a list of what it can do with the implication being that it can't do anything else. This is nonsense. If you can think of a way of doing something then you can build an expert system to do it.

Someone else's expert system may only be able to do such-and-such. But yours! That's another matter. Your expert system is under no such constraints. Yours is going to be bigger and better than anyone else's (if that's how you build it, that is). It is presumptuous of anyone to tell you what your expert system can't do.

Take for instance:

"AN EXPERT SYSTEM CAN ONLY BE EXPERT ON ONE THING." Patently untrue. All you've got to do is to build one that's expert on two things and you've proved the critics wrong.

"AN EXPERT SYSTEM CAN ONLY DO WHAT A HUMAN EXPERT COULD DO (AT BEST)" Also untrue. Suppose you chose to build a system for which there aren't any human experts? If it ever works at all you've proved this statement false. More typically, there are plenty of fields in which human expertise is less than complete. In such a case, all your expert system has to do is be a little bit better than normal human judgement.

"EXPERT SYSTEMS WILL NEVER REPLACE MAN." Of course they will. There'd be no point building them if they couldn't.

Human nature being what it is, it's always possible to find someone who will disagree with this book's description of expert systems. You may even disagree with it yourself. So, let's note the following points:

To some people, a special characteristic of expert systems is that they are 'adaptive' - meaning that the behaviour of the program changes (usually for the better) over a period of time. They can do this either by holding their knowledge in rules which can very readily be altered by the user, or by building up their knowledge from their own analysis of the inputs with

little intervention on the part of the user. In both these senses the systems described in this book are adaptive.

An aim of many of today's expert systems is to make them generally expert in a wide range of tasks simply by drawing a distinction between the knowledge they use and the mechanisms which manipulate that knowledge. By changing the nature of the specific knowledge they use, the expert system then becomes able to exercise its manipulative ability to become expert in a new field. For instance, an expert system designed to carry out medical diagnosis might, by removing the medical knowledge and substituting knowledge pertaining to structural engineering, become expert in structural engineering instead. The programs described in this book are organised in this way.

It is often said that a special characteristic of expert systems is the rule-based organisation of their instructions. Unlike conventional computer programs which proceed in a line-by-line system of execution, expert systems consist of a collection of rules which are not executed sequentially but which 'fire' only as and when appropriate conditions are met. In the literal sense this description is rather misleading because, apart from some very rare machines, all computers step through their code in much the same way. Conceptually, however, the point is a reasonable one to make - but only in the sense that expert systems often behave as if every line of code began with the word IF.... . And that, indirectly maybe, is how the programs given in this book work.

Chapter 2

A Statistical Scheme

2.1 Setting up a matrix

But enough, you may well exclaim, of preliminaries. Having been promised an expert system, where is it? How, in short, do you build your own expert system?

Well, it's easy.

First of all define a two dimensional array and think of it as a rectangular matrix. Think of the questions you want to ask your expert system and label the columns of the matrix with all of the possible answers. Then think of all the pieces of information which the expert might need in order to arrive at these answers and label the rows of the matrix with these pieces of information.

Take a concrete example, to make it clearer.

Suppose that you'd like your own expert system to tell you if it's going to rain tomorrow. The idea is that you want to be able to sidle up to your computer and ask: Is it going to rain tomorrow? And the computer will pause for thought and reply: Yes or No. To avoid confusion it won't reply: Yes and No.

So, there are two possible answers. Therefore we want a matrix with two columns. We don't at this stage know how many pieces of information the expert will need in order to answer the question but let's allow, say, ten rows in the matrix. So, in BASIC, DIM E(10,2). In case you hadn't guessed, E stands for Expert. Graphically, we have:

Fig. 2.1
An expert's matrix

Observation	Rain tomorrow	No rain tomorrow
1	a	b
2	c	d
.	.	.
.	.	.
.	.	.
10	s	t

Where, down the left hand side, we've listed the pieces of information available to the system to make a decision about the weather. These are the numbers, say, 1 to 10 and might correspond to questions such as: Is it raining today? Is it cold today? And so on. The letters a, b,.....s, t in the array are items which, as yet, we do not have but they will represent the possibility of each outcome being true. They will later be filled out with information relating to the weather. The task for the expert system is to decide which of the two columns, Rain or No Rain Tomorrow, it should select given the answers to the questions 1 to 10.

For example, observation 1 may occur when the answer is YES to the question
"Is it cold today?"
From numerous observations (by experts) of cold days, there is a 60% chance that it will rain tomorrow. So, 'a' will be 60 or 0.6 if you like decimals. In a like manner, we would obtain values of b, c and so on, ensuring that we could get a sensible sort of answer based on the observations we make.

Now, as it stands, things might be looking a bit more concrete than they were but, despite that, have we really done anything useful? Well, yes. We've established a Knowledge Base and a Domain of Enquiry both of which are good phrases to remember next time you feel like waxing eloquent in a local hostelry.

Take the Domain of Enquiry. Well, that's what it's all about. Literally. It's the subject matter of the expert system. If something falls within the Domain of Enquiry you can ask your expert system about it and, if it isn't, you can't. Simple as that, really. In this case the Domain of Enquiry is the

weather (or rain, if you want to restrict things a bit). There's absolutely no point in your asking this expert how you can become a millionaire because such a question would be outside its Domain of Enquiry. There may be other reasons why you can't ask that question but one will do for the moment.

Now consider the Knowledge Base. In this case, it is the array E(10,2). It contains all of the knowledge the expert has on the subject and, in this case, significantly, it's empty. It doesn't know a thing about the weather. Which means that, so far, there's absolutely no point in your asking this expert about the weather either.

As this is so crucial to our main theme, take a look at Figure 2.2

The Domain of Enquiry is the 'field' in which the Expert System is intended to be expert. In this example we have a weather-predicting expert so the Domain of Enquiry is The Weather.

To be expert in this field, the expert system needs a Knowledge Base. That is the information which we give it about the subject of weather. In an ideal world (which, in our case, we do not have) the Knowledge Base would encompass the whole of the Domain of Enquiry i.e. it would know everything there was to know about the subject in hand (the Weather). In practice, it's unlikely to know everything - it will just know a portion. We can represent this by showing the Knowledge Base as being smaller than, but existing within, the Domain of Enquiry.

When a specific question arises, if asked with any hope of an answer, it must fall within the Domain of Enquiry. However, it may or may not fall exactly within the Knowledge Base on our diagram. Exactly where we place a specific example depends on the extent to which the Knowledge Base within the Expert System encompasses the specific example we produce.

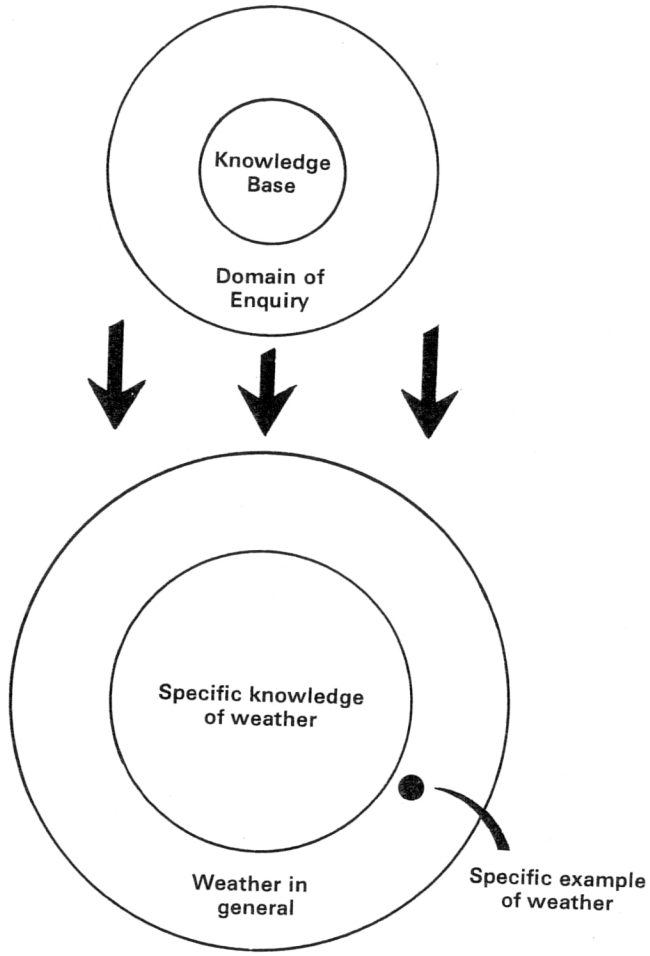
Consider a human expert to whom you turn with a thrusting question concerning the likelihood of rain tomorrow. Most people would be willing to hazard some kind of guess on the matter. But, they have an advantage over the computer to date: they know what weather is and they can look out of the window to see what today's weather is like - which might give them a clue about tomorrow's weather. Obviously, we want our expert system to do a similiar kind of thing.

So, let's distinguish two different kinds of information in the problem. We could call them a number of different things, but let's try Fixed Information and Variable Information.

Fixed Information is what would go in the Knowledge Base. It contains invariable data which, in this case, would be about the weather in general.

Variable Information is what wouldn't go in the Knowledge Base. This is specific information relating to the problem in hand. In this case it would be the specific information which would enable the expert to say if it would rain tomorrow or not given that we are asking the question today and not just asking the question about any day in general.

Fig. 2.2
Knowledge base and domain of enquiry



In a way, these two sets of information are like the difference between programs and data. The fixed information is part of the program and the variable information is the data for this specific problem. But, just to make sure that the subject doesn't become too clear-cut, it's worth making it woolly again by pointing out that we might want to alter the fixed information quite often (so it's not really fixed after all) and it might even be as variable as the variable information at times. This is because most experts change their methods of working as time progresses and we wouldn't want to prevent our expert system from learning as time goes on, would we? In fact, this is often a feature of expert systems.

Anyway, back to the expert on weather. We've just asked if it's going to rain tomorrow and received a reply to the effect that it hasn't a clue. What next?

Well, give it something to work on. Tell it something about the weather. For instance: if it rains today it's more likely to rain tomorrow. This is because rain tends to go in spells. We have wet spells and dry spells. Therefore if it's raining today, it's likely to be a rainy spell which implies that it will rain tomorrow. That, incidentally, is more or less true. So what we're doing is embodying a bit of human expertise in our expert system.

Suppose we reasoned that, if it's raining today there's a 60 per cent chance of it raining tomorrow. Therefore, by an amazing feat of arithmetic we see that there's a 40 per cent chance of it being dry tomorrow. Further suppose that if it's dry today there's a 55 per cent chance of it being dry tomorrow. By the same feat of calculation there's a 45 per cent chance of it's being wet. Now re-draw our original array with these gems of knowledge built in:

	1. Rain tomorrow	2. No rain tomorrow
1. Wet	60	40
2. Dry	45	55

At this point it becomes quite obvious what we should do. The expert system has to print out the statement Rain Tomorrow or the statement No Rain Tomorrow. To decide which, it asks if it is wet today. If it is, it prints out Rain Tomorrow because that's the most likely outcome. If it isn't wet today it can do one of two things depending on how you want to program it.

It can either print out No Rain Tomorrow on the grounds that if it isn't raining it must be dry and, therefore, it's most likely to be dry tomorrow. Or

it can specifically ask if it's dry today and, on receiving the answer Yes, proceed from there.

Now, all of this might be concrete but, is it useful?

After all, you can always listen to a weather forecast or, failing that, just wait and see what the weather does. It's not really that important, you might feel.

And, what's more, you probably haven't got much faith in those numbers we put in the matrix (sorry, Knowledge Base) in the first place.

Well, it's fair enough to have some reservations. But consider some aspects of what we've done.

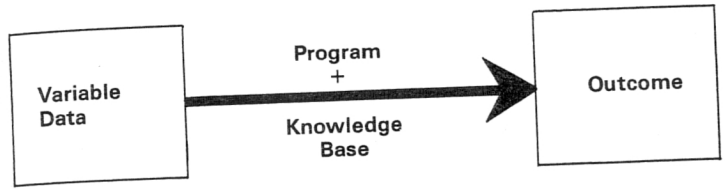
Take those numbers we dreamed up. They are, certainly, a bit ad hoc. But suppose they hadn't been. Suppose that they had been precisely determined numbers - what then? Well, it would have made some difference in some circumstances. Say, one of the numbers had been 100 per cent and another 0 per cent. Then our expert could have made a certain decision, not just a likely guess. Further, suppose that we didn't know what those numbers had been at all. We might not have known that rain typically precedes rain or that weather goes in spells. If our expert system could have worked that out for us, then it would have known more than we did originally and it might have been better at weather prediction than we are.

Now consider the actual question asked: Is it going to rain tomorrow? Probably rather trivial for the super-smooth expert system with which we intend to impress the world. But it could have been a different question. The rows and columns of the matrix could have been labelled differently providing different variables, different outcomes, and a different field of expertise.

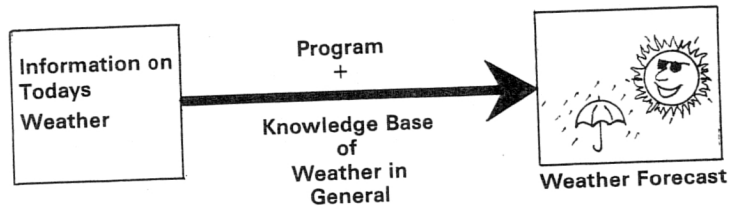
We might have asked: Is Greyhound X going to win in this race? Or: Does my headache denote a hangover?.

Providing that we can contrive to specify some variables, some outcomes, and some method of linking the two, we can tackle a wide variety of problems with our expert system. The trick is simply to have some method, like this, for getting the system organised around the problem to start with.

Fig. 2.3
Weather forecasting



specifically: –



2.2 Probabilities

In the previous example we thought of an event as being, say, 55 per cent likely to occur; 45 per cent likely to occur; 100 per cent likely to occur. There's no particular reason why we shouldn't continue to do that but the use of probabilities has some advantages.

Probabilities are very precise things. They are represented by numbers which range between 0 and 1.

If an event is absolutely certain to occur, then it has a probability of occurrence equal to 1. If an event has no chance whatever of occurring then it has a probability of occurrence equal to 0.

If an event has a fifty:fifty chance of occurring then it has a probability of occurrence equal to .5.
All other cases fall somewhere in between.

Consider the cases of 0 and 1. There is a difference between a probability of exactly 1 and a probability which is approximately 1 - so close to 1 that we may as well write it down as 1. If an event has a probability of exactly 1 then it *must* occur. It simply cannot not occur. It may have a probability number associated with it but there is no element of chance involved with it at all. It almost has the force of a causal statement. Consider: if you leave go of your glass when there's nothing underneath it then it will fall to the ground. That is a certain event and has probability 1 associated with it. It's also a causal statement in as much as the falling glass is caused by the releasing of your grasp.

But consider: when you eventually leave that bar in which you had a glass you'll find that everyone else leaves as well. You all, in short, walk out of the public house at the same time with probability 1 - it is a certain event. But this isn't a strictly causal event. It isn't the fact that you, personally, are leaving that causes everyone else to decide that the hostelry holds no further pleasures for them. It is the fact that the barman is closing down for the night and is throwing you all out that is the causal item.

So, if two or more events occur together with probability 1 the relationship between them need not be strictly causal - they might just happen to always go together. But they still happen together and there's no maybe about that.

Now go back into the bar and drop that glass again.

We said it was going to fall to the ground - but we didn't say it was going to actually break. It probably will and, maybe it will break with probability .999999. In fact, to save writing out all those 9's we could say that it will break with probability approximately 1. But this isn't the same as saying it will break with probability exactly 1. If the probability had been exactly 1 then the glass would have broken - and no two ways about it. But as long as this magic figure of 1 is not exactly achieved there will always be some, albeit small, probability that the event won't occur - and the difference could well be an unbroken glass.

The same arguments go for probability 0. A probability 0 event *can't* occur. But an event which has a probability of approximately 0 can certainly occur even though it's very rare.

By now, it may have dawned on you that there's no particular reason why we shouldn't have used percentages. 100 per cent likely to occur means that you're quite certain about something. For less certainty, you might be 99.99 per cent certain and so on.

You can use any scale you choose really. You could have a scale from minus five to plus five if you wanted. Or anything. The only reason for labouring the point is to make clear that it's possible to talk about probabilities in a very precise fashion. For our purposes a completely causal event can be described by probability numbers if we want to describe it like that. And the reason for using the usual 0,1 range of probabilities is really to remind ourselves that what we are doing is precise.

Take our statement that "if it rains today then there's a 60 per cent chance of it raining tomorrow." We mean, of course, a probability of occurrence of 0.6 for the event. Now, from this we can deduce that if we watched an infinite number of rainy days we would find that 0.6 of them were followed by another rainy day. There is no if, but, or maybe. In this case, of course, our figure is almost certainly inaccurate but that doesn't render the statement in which it was used meaningless. The meaning was very precise. Only the figure was wrong. And, if we'd got the figure right, it would not only have been meaningful, it would have been true!

That's the reason for stressing the point.

Imagine yourself holding forth about your own expert system. "It decided," you announce, "that there was a .75 probability of World War Three starting on any one day of next week."
"Oh," responds a bored audience. "It isn't sure about it then?"

And they start talking about something more certain. Like the weather, for instance.

It could be absolutely infuriating to have an expert system which could work out something like that and then have it dismissed so lightly. Really sends the blood pressure up, that sort of thing.

But listen to what you said.

You were using a probability measure, so, if there's a .75 probability of World War Version 3 starting on any one day of next week then the probability of World War 3 not starting on any one day of next week is 1 - .75 = .25. And, for War to hold off right through the week, not occurring on each and every one of its seven days, we have a probability of .25(power of 7). So the probability of a Peaceful Week is 0.000061035.

The probability of not having a Peaceful Week is, therefore, 1 - .25(power of 7) = .999938965. Which is the probability of World War Three breaking out on some day next week.

This all assumes, of course, that when World War 3 does break out it does so quite independently of anything which might have occurred on previous days but, as it's all computer-controlled nowadays, this doesn't seem to be an unreasonable assumption to make - after all, chips do tend to go faulty in a statistically-independent fashion!

You haven't specified which day of the week it's all going to happen but there would be less than 6 chances in 100,000 that it wouldn't happen at all. So, all in all, everyone could reasonably go out and buy tin helmets on the strength of the information you've just given them.

But the important thing to remember is that you needn't look down on probabilities. You can calculate a lot of exact things with them and make some very exact statements, some of which just might be true.

And, on a more practical note, if you're going to build your own expert system then you won't be inspired to get very far with it if you think that anything with a probability in it is only a vague 'chance' and not really worth considering seriously at all.

2.3 More probabilities

Now, it could be that probabilities are, as it were, an open book to you.

On the other hand, it might be worthwhile, just in case they aren't,mentioning a few basic points about what happens when you have more than one probability.

Some notation might help.

We'll try:
P(A), P(A&B), P(A:B) and define them this way:

P(A) is the probability of A occurring.
P(A&B) is the probability of both event A and event B occurring.
P(A:B) is the probability of event A occurring given that event B has occurred.

Respectively, we say that:

P(A) is the probability of A
P(A&B) is the JOINT probability of A and B
P(A:B) is the CONDITIONAL probability of A given B.
They are not different ways of saying the same thing.

Going back to our weather example, we had the matrix (using probabilities, rather than percentages):

	1. Rain Tomorrow	2. No Rain Tomorrow
1. Wet	. 6	. 4
2. Dry	.45	.55

These are CONDITIONAL probabilities. They are the probabilities of particular weather tomorrow GIVEN today's weather. For instance, there is probability .6 of rain tomorrow if there's rain today. That is not the same as either the probability of rain tomorrow, or the JOINT probability of rain tomorrow and rain today. At a first reading it might sound as if they are much the same thing. But they aren't and it's worth sorting the matter out fairly early on otherwise all that stuff about probabilities being precise, exact things tends to be wasted.

First, what's the probability of rain tomorrow?

Note that we haven't said anything about today's weather. It's a single item in isolation of the previous day's weather. Just, in general, will it rain tomorrow? The information we have is that if, on any day, it's wet then the probability of rain tomorrow is .6. On the other hand, if it's dry, the

probability of rain tomorrow is .45. What should we do to get the overall probability of rain tomorrow? Add them up (l.05)? Average them (.525)?

Take P(R) as the probability of rain tomorrow and use W and D to mean Wet and Dry today respectively. We want to find P(R) and our table gives us P(R:W), P(R:D) in the first column.

At this stage we introduce a formula:

P(A&B) = P(A:B)P(B).
In words this says that the probability of both A and B occurring is equal to the probability of A given B, multiplied by the probability of B.

If you think about it you'll find it's fairly reasonable. We want to know about A and B occurring. If B definitely occurs then there is a probability P(A:B) that A also occurs. But B only occurs with probability P(B) so we have to allow for that to get the figure for both A and B occurring. Hence we multiply P(A:B) times P(B) which, as P(B) is no greater than 1 tends to make the answer smaller, allowing for the chance of B not occurring.

So, the probability that it will rain tomorrow and that it rains today is:
P(R&W) = P(R:W)P(W)

i.e. the probability of rain tomorrow and today is equal to the probability of rain tomorrow given rain today, times the probability of rain today.

Or: P(R&W) = .6 P(W)

And, for a dry day today:
P(R&D) = P(R:D)P(D)

Or: P(R&D) = .45 P(D).

But we wanted P(R), the probability of rain tomorrow. Well, the probability of rain tomorrow is obviously equal to the probability of rain tomorrow and rain today plus the probability of rain tomorrow and a dry day today. So:
P(R) = P(R&W) + P(R&D) = P(R:W)P(W) + P(R:D) P(D)
= .6 P(W) + .45 P(D)

At this point we recall that today is either a wet day or a dry day.
So: P(W) + P(D) = 1, it must be one or the other with probability 1.

And we get

P(R) = .6 P(W) + .45 (1 - P(W))
= .6 P(W) + .45 - .45 P(W)
= .15 P(W) + .45

So, in order to answer the question: what is the probability of rain tomorrow? we need to know the probability of a wet day today.

It is, of course, a question that we can answer easily because we reckon that the probability of rain tomorrow is the same as rain on any other day which means that it's the same as the probability of a wet day today.

That is: P(R)=P(W)

So: P(R) = .15 P(R) + .45
.85 P(R) = .45
P(R) = .45/.85
= .53 approximately.

Or, more likely than not, it will rain tomorrow as it has done today and will do until the world ends. That's our weather for you.

Note that we couldn't have got this result if P(R) hadn't been the same as P(W). If P(W) had been, say, mist today this wouldn't have been the same as rain tomorrow. Then we'd have had to go to some other source to find out a value for P(W).

The important point to note is that the probability of rain tomorrow isn't sitting in that original table staring at you. Those are CONDITIONAL probabilities, not probabilities of isolated events, and they aren't the same thing at all.

In the course of working all that out we also obtained some JOINT probabilities, P(R&W) and P(R&D), which were .6 P(W) and .45 P(D),respectively. Now, we know that P(R) = P(W) and that P(D) = 1 - P(W) = 1 - P(R).

So: P(R&W) = .6 P(R) = .3176 approximately and
P(R&D) = .45 (1 - P(R)) = .2117 approximately.

These probabilities are much lower, because we're asking what the probability of rain tomorrow and rain today is (ie. two wet days in succession) and what is the probability of rain tomorrow and a dry day today (ie. two days of alternating weather). This joint probability makes a more exact request for a specific situation. Less is assumed and the chances of getting just what we ask for are so much less.

In all probability, you're fed up with probabilities by now. Certainly they don't make particularly interesting reading owing to a pretty complete absence of plot, corpses and jokes.

There is one more thing to be done in this section though. After that we'll get back to the expert system proper.

On July 16, 1917 no less than 4.65 inches of rain fell in 2.5 hours at Kensington. No less than 8 inches fell in 5 hours near Bridgewater on August 18, 1924. On June 28, 1917, 9.56 inches of rain fell during one day at Bruton, Somerset.

Moving to other parts of the Empire:

In Gibralter on October 25, 1836, 30.11 inches of rain fell in a single day.

In Bagnio in the Philippines on the four days July 14 to 17, 1911, the rainfall was: 35 inches, 29 inches, 17 inches and 8 inches.

All of which goes to show something we haven't built into our expert system yet: the knowledge that sometimes it never rains, it pours.

2.4 More variables

It is a typical day again. It is cold, wet, windy and foggy. You turn up the collar of your threadbare jacket against the elements and light a soggy cigarette to warm yourself about its glow. A raindrop of more than usual size extinguishes the cigarette for you.

Will it, you ask yourself, be like this tomorrow?

Well, of course, it probably will. Anyone could tell you that, for this is England, the land where anything is possible as long as it involves getting rain down the back of one's neck.

But, just to indulge a wild fantasy, it has to be admitted that a dry day might occur. So, will it rain tomorrow?

Switch on the expert system to find out. Before switching on, check for any rainwater inside the casing. Remember matrix E(10,2)? Well, fill it in as follows:

	1. Rain tommorrow	2. No rain tommorrow
1. Cold	P(Rain:Cold)	P(Dry:Cold)
2. Wet	P(Rain:Wet)	P(Dry:Wet)
3. Windy	P(Rain:Wind)	P(Dry:Wind)
4. Foggy	P(Rain:Fog)	P(Dry:Fog)

Say that you know the conditional probabilities in the matrix and can fill them in easily. Now if, say, it was Cold today and NOT Wet, Windy or

Foggy then you'd be laughing (in a manner of speaking, that is). Because all you'd have to do is to look at the table and see whether the conditional probability of Rain given Cold was greater than the conditional probability of No Rain given Cold and choose whichever was the most likely outcome. Go for the biggest probability.

The question that really arises though is what to do when more than one variable occurs. In this case we have four variables. How will your expert system work out if it will be wet or dry tomorrow?

It may sound a trivial question. After all, if it really is cold, wet, windy and foggy then it sounds as if any idiot could predict rain with a fair degree of certainty. It sounds like a bad spell of weather with more to come. And that, really, is just the point.

Fig. 2.4
Two correlated variables

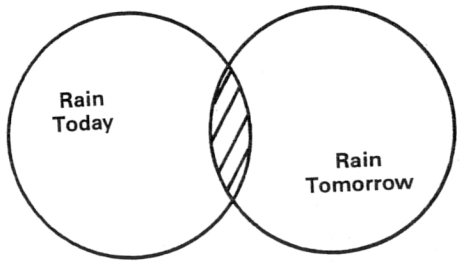
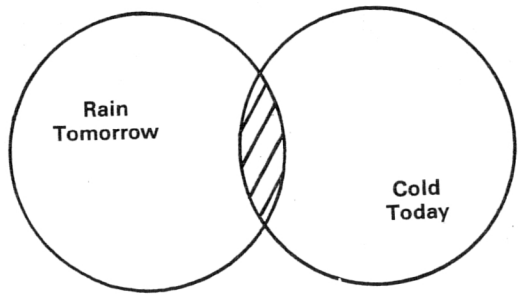


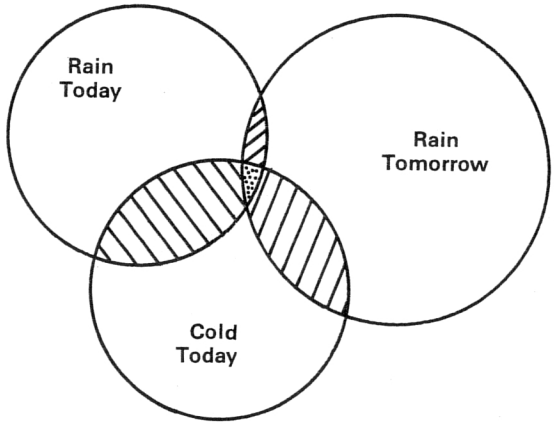
Figure 2.4 represents two variables. Rain Today and Rain Tomorrow, and the shaded area shows that these two variables are correlated. If we know whether or not we have Rain Today we can judge the probability of Rain Tomorrow by looking at the extent to which these two variables 'overlap'.

Fig. 2.5



Similarly, we can do the same if one of our variables is Cold Today, instead.

Fig. 2.6
The 3 variable case



However, if we have information on both Rain Today and Cold Today simultaneously the problem becomes much more difficult because the size of the overlap into Rain Tomorrow depends on the overlap that exists between Rain Today and Cold Today i.e. the extent to which these two variables are correlated. This is shown in Figure 2.6

If even an idiot can make a decent guess on the basis of that information then surely your expert system ought to be able to make at least as good a guess.

First, let's think of the variables we have:

It's cold. That sounds like bad weather - but does it mean rain? Actually, it often doesn't. In cold weather the atmosphere can't contain so much water vapour as in warm weather so, if it did rain, there wouldn't be so much water in the sky to come down. In defence of this point you could look at monthly rainfall figures for Great Britain. You'll find that it doesn't rain much more in winter months than it does in summer months - which can be a bit of a surprise if you have strong memories of getting drenched in winter. Probably, you just don't notice warm rain so much as cold rain.

It's wet. Well, weather tends to go in spells, so a wet day could well precede a wet day. Unless, of course, there wasn't any left up there to come down tomorrow. (This latter suggestion is known as Optimism.)

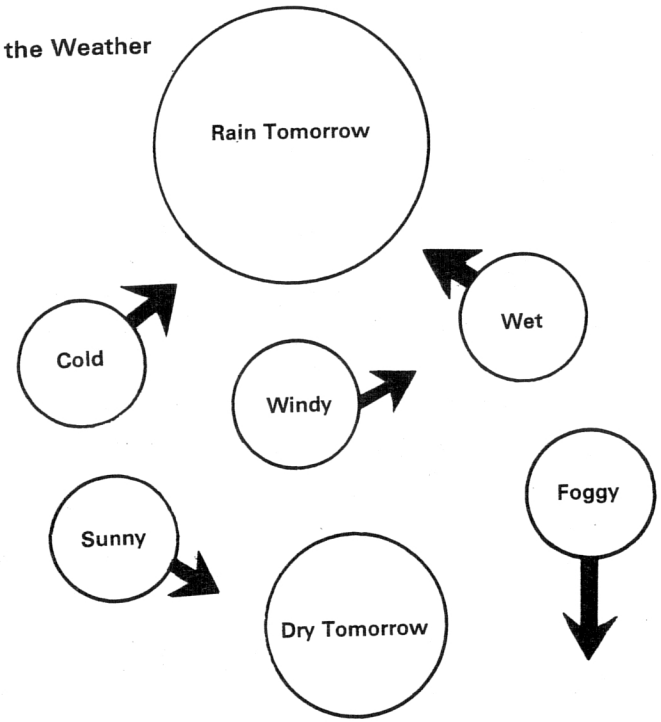
It's windy. This often precedes rain. Warm, moist air being blown into a cold area will rise, cool, and pour water down onto our heads. And movements of air, like this, are just another way of describing a windy day.

It's foggy. Well, if it's foggy it's not likely to be windy,is it? Fog usually arises from still, damp air. It may or may not turn to rain.

So: taking our four variables, what do we have? Nothing, at this stage, very certain. Some but not all of them point to rain. Certainly, it would be handy if they all did indicate rain with a probability greater than .5 because, if they did, we'd have no hesitation in getting our expert system to predict rain. But suppose that instead of Foggy we'd used the variable Sunny. And just suppose that the weather had been: cold, wet, windy and sunny. Now this would be what meteorologists call 'unsettled weather' - but, for our purposes, it's just a nuisance because Sunny tends to indicate a dry day tomorrow whereas all of the other variables tend to indicate rain.

In general, we have a set of pointers and we're alright as long as they all point in the same direction.

Fig. 2.7
Pointers to the Weather



If we have several indicators, we can get our expert to judge the correct outcome easily if all the indicators point in the same direction. If they all point in different directions, problems arise because we don't know how much reliance to put on any one indicator.

So: what to do about it?

Back to our conditional probabilities and we started off with P(Rain:Rain today) - the probability of Rain given that it's raining today.

But what we've got is:

P(Rain:event X) where the event X is the event that today it is cold, wet, windy and foggy.

What we want to know is:

P(Rain:Cold & Wet & Windy & Foggy)

and specifically, we want to know whether it's greater or smaller than:

P(No Rain:Cold & Wet & Windy & Foggy).

Going back to one of the formulae we had before:

$$P(\text{Rain:event X}) = P(\text{Rain \& event X}) \,/\, P(\text{event X})$$

or:

$$P(\text{Rain:Cold \& Wet \& Windy \& Foggy}) = \frac{P(\text{Rain tomorrow \& Cold today \& Wet today \& Windy today \& Foggy today})}{P(\text{Cold \& Wet \& Windy \& Foggy})}$$

Now, at this point, it should have become a little clearer why we spent some time talking about different kinds of probabilities earlier on. It was to make clear that there are different kinds of information which we can have about a situation and different kinds of questions we can ask.

For instance, we started off with statements like: P(Rain:rain). Which isn't in exactly the same form as the question we've just asked. You can't, in general, take a lot of simple statements about conditional probabilities and turn them straight into one big statement about the probability of one event given the joint occurrence of a lot of other events.

In case you're not convinced, consider an example.

It's misty and it's foggy. The probability of rain the next day given mist today is .75, likewise the probability of rain given fog. As it's misty and it's foggy we add the two together and get a probability of rain tomorrow of 1.5 which is obviously wrong. Probabilities can't exceed 1.

But suppose we consider P(A&B), the probability of two events together. Let the first event be rain tomorrow given mist today. Let the second event be rain tomorrow given fog today. Suppose we could say the events are independent of each other then P(A&B) = P(A)P(B) which is equal,in this case, to .75 times .75 = .5625.

Well, that's worse. With both events pointing at rain we've calculated a probability of rain actually less than either pointer by itself!

The answer is, of course, that mist and fog are the same thing. They're both caused by water droplets in the air and the only difference is that fog has larger water droplets than does mist.

So fog and mist are just different words for the same thing and by including both statements we've said nothing extra about the situation at all.

Hence, the probability of rain tomorrow is .75 - the same as predicted by either statement.

And it only requires brief thought to see that, in our original table there was absolutely nothing to indicate this. If we had Mist and Fog as entries on the left-hand side with conditional probabilities next to them we would, effectively, have had the same entry twice - but with nothing to indicate to the expert system that this might be so.

To some extent this problem is going to occur in lots of different ways for other variables because what we've found is a correlation amongst the variables.If they were all independent of each other then life would be easier but, in general, they won't be. Consider Wet & Windy. Now, these two variables certainly don't mean the same thing at all. But if it's wet it's often windy too - they aren't independent. If we had included Sunny then that wouldn't have been independent of Foggy either because when it's one it tends not to be the other.

Happily, there is a way of dealing with this problem. Unhappily, it tends to involve some labour. What you do is this:

1. Think of all of the variables you have, say n.

2. Think of all the possible combinations of these variables. This is the sum 'n pick x' for x=0 to n

i.e.

$$\frac{n!}{(n-0)\,!0!} + \frac{n!}{(n-1)\,!1!} + \frac{n!}{(n-2)\,!2!} + \ldots\ldots + \text{and so on}$$

In the case of four variables on the weather, we have:

$$\frac{4!}{(4-0)\,!} + \frac{4!}{(4-1)\,!1!} + \frac{4!}{(4-2)\,!2!} + \frac{4!}{(4-3)\,!3!} + \frac{4!}{(4-4)\,!4!}$$

which is:

$$1 + 4 + 6 + 4 + 1 = 16$$

i.e. there are 16 different combinations which could arise with our four Yes/No variables.

The exclamation marks mean 'factorial' as well as being rather like a cry of computational anguish.

3. Make out a new list of variables. This time you list all of those combinations of variables as well as the variables themselves.

4. Assign a probability for each outcome given the occurrence of each of the specific combinations of variables.

5. When you want to use the system you just look to see what combination of variables you have, find that combination in your extended table, and read off the probability for each of the possible outcomes.

6. Having contemplated this you raise an eyebrow, purse your lips, furrow your brow and exclaim: "You must be joking!" (Exclamation mark, not factorial).

Well, yes, sorry about that. But that's how you do it. It is a bit time-consuming one does admit....

It would be much easier, of course, if you didn't have so many variables. It's really your own fault for thinking you can just sling anything at a computer simply because it's got a lot of memory.

2.5 Bayes' Theorem

If however, you do try to make a go of this method you could always confuse yourself further by introducing Bayes' Theorem in your calculation. The Reverend Bayes, as you can guess from his title, devoted his life to the study of things eternal like, for instance, problems in statistics. And his theorem states that:

$$P(R:X) = \frac{P(X:R)P(R)}{P(X:R)P(R) + P(X:\text{not } R)P(\text{not } R)}$$

In other words, consider R to be the event Rain Tomorrow and X to be a particular combination of events describing today's weather (say, wet, windy, cold and horrid).

Then, the probability of Rain tomorrow given X today is equal to the probability of X today given that it's going to rain tomorrow times the probability of rain tomorrow, divided by the total probability of X occurring anyway.

If you're not happy about why this should be true consider:

$$P(R:X) = \frac{P(R\&X)}{P(X)}$$ this is a standard result we gave earlier.

$$\text{So, also } P(X:R) = \frac{P(X\&R)}{P(R)}$$

$$\text{So, } P(X\&R) = P(X:R)P(R) = P(R:X)P(X)$$

$$\text{So, } P(R:X) = \frac{P(X:R)P(R)}{P(X)} = \frac{P(X:R)P(R)}{P(X:R)P(R) + P(X:\text{not } R)P(\text{not } R)}$$

Now, if you knew P(R:X) you could just enter it in at the start - no trouble. But you might find it easier to find P(X:R) - it depends to some extent on the nature of the question as to which is easier. For the one question you have to provide the probability of rain tomorrow given X today. For the other you have to provide the probability of X today given that you know

it's going to rain tomorrow (or, more realistically, the probability of X yesterday for each day that it rained today). You also (just so that you don't feel you're getting on top of the problem) have to give P(X:not R) the probability of X given that it's going to be a dry day today. And you might well feel that this method is even worse than any previous methods.

Which isn't quite true. It's certainly pretty bad if all the events are correlated with each other so that you have to provide probability values for each and every possible combination. But if they are independent then you can calculate an overall P(X:R) much more easily than you can calculate an overall P(R:X).

Suppose, for instance, that X is the pair of events Y and Z.

Then: P(X:R) = P(Y&Z:R) = P(Y:R)P(Z:R) a simple multiplication.

But: P(R:X) = P(R:Y&Z) does not equal P(R:Y)P(R:Z). If you don't believe this, figure the two calculations out separately and see what sort of different results you get.

Later (much later, in the second half of this book) there's a program given which uses this method to build an expert system. It works by assuming that all events are, in fact, independent of each other - if they weren't it would run badly. Often, systems are designed which assume independence of the variables just to make life easier. Certainly it eases the problem of that vast number of combinations which must otherwise be considered but often the asumptions of independence are only roughly true. And if there is a lot of intercorrelation among the variables the problem really becomes immense.

If ever you get to the end of this book maybe you ought to come back to this section and think about it all again when you've seen how a system might be implemented using this particular method of working. Or, maybe, when you get to the end you could just sell the book and spend the proceeds on drink. That way you become less likely to care whether or not it's going to rain tomorrow.

Chapter 3

Avoiding Probabilities

By this time a strange unease should have filled your very being. You thought that building an expert system would be easy. In fact, you were more or less promised that it was. You have a computer - why can't that do some of the work for you? What, you ask, is all this talk of probabilities? Why, for instance, do you have to tell the expert the probability of every little outcome in order to get a decision out of it?

Frankly, if you've got to do all of this work then you hardly need a computer. You might as well save a bit of time and do it yourself.

And, apart from anything else, if you knew the probabilities associated with each possible set of events you wouldn't need a mechanical expert. The reason you want an expert system is to do things for you which you can't do for yourself. To tell you things which you didn't previously know.

All of which are fair comments, one feels. But, your very own expert system is different to most existing expert systems in this way:

Most expert systems to date rely on fairly intensive research which, roughly speaking, involves picking the brains of an expert to find out the odds on each and every possible outcome to each and every possible event. The (human) expert knows these odds. People that pick his brains and write an expert system then know the odds too. They then write them into the expert system so that it knows all the odds after due programming. They then give the expert system to someone who *doesn't* know all of the odds and that person is then duly impressed.

Some expert systems work slightly differently in that the computer picks the expert's brains - but in almost every case the presence of a (human) expert in the subject is assumed.

But consider your case as you build your very own expert system. If you

knew everything about the problem, you wouldn't need an expert system to tell you about it. Typically, you don't know all there is to know. And, as you don't know it, you can't program it. And, if you went to all the trouble of finding out everything about the subject in which you required expertise it would then almost seem like a lot of additional trouble to write it into a program instead of just making a mental note about it.

In short, your requirements for an expert system are very different to the requirements of those who have built such systems to date.

You want something that's fairly quick and easy to build, gives good results, and doesn't involve you in taking up a lifetime's study of the subject in which expertise is needed.

So that's what we'll do. As outlined above. Right at the beginning we said that your expert system would be better than anyone else's - or, at least, different. This is largely where the differences start to appear.

3.1 How to make the computer do the hard work

The system now changes. You don't give the machine probabilities anymore - although you do know enough about probabilities to appreciate what it's got to do. You'll just give the machine enough information so that it can work out what to do by itself. It's not only going to have to be an expert system. It's also going to have to be a learning system because it's the machine that's going to have to do the learning, not you. That is what is called Delegation.

And it's also about time that you had a program which you can put on the computer so that you feel that you're getting somewhere.

What we'll do is this: we'll set up the expert system so that it can have a training session during which it can learn to make a decision on the basis of experience. After it's been trained we let it loose and allow it to make real decisions using its expert judgement. Unless you examine the program carefully, you won't need to know how it came to its decisions at all. Certainly, you won't have to tell it explicitly, which is what you would have done previously.

Hopefully, this will make you feel a bit more optimistic about the whole business.

3.2 The learning system

The program that we will use is listed in Fig. 3.1 and a flowchart for it is shown in Fig. 3.2. It is written in BASIC for the Apple II and Sinclair Spectrum.

Fig. 3.1
A simple learning program

Apple II listing

```
10   HOME : INPUT "HOW MANY VARIABLES HAVE YOU ?";V
20   DIM V(V),R(V),V$(V)
30   FOR I=1 TO V: v(I)=0 : R(I)=0: NEXT
40   PRINT "PLEASE NAME THESE VARIABLES"
50   FOR I = 1 TO V: INPUT "VARIABLE NAME ";V$(I): NEXT
60   PRINT "PLEASE NAME THE OUTCOMES"
70   INPUT "OUTCOME 1 ";Q1$
80   INPUT "OUTCOME 2 ";Q2$
90   FOR I = 1 TO V:V(I) = 0
100  PRINT "VARIABLE ";V$(I)
110  INPUT "IS THIS VARIABLE THE CASE ?";A$
120  IF A$ = "Y" THEN :V(I) = 1
130  NEXT
140   D = 0
150  FOR I = 1 TO V
160  D = D + V(I) * R(I)
170  NEXT
180  IF D > = 0 THEN : PRINT "OUTCOME IS ";Q1$: INPUT "IS THIS RIGHT ?";A$: IF A$ =
     "Y" THEN : GOTO 90
190  IF D < 0 THEN : PRINT "OUTCOME IS ";Q2$: INPUT "IS THIS RIGHT ?";A$: IF A$ = "Y"
     THEN : GOTO 90
200  IF D > = 0 AND A$ = "N" THEN : FOR I = 1 TO V:R(I) = R(I) - V(I): NEXT
210  IF D < 0 AND A$ = "N" THEN : FOR I = 1 TO V:R(I) = R(I) + V(I): NEXT
220  GOTO 90
```

Sinclair Spectrum listing

```
  10 CLS : INPUT "How many varia
bles have you?";v
  20 DIM q$(20): DIM r$(20): DIM
 v(v): DIM r(v): DIM v$(v,20)
  30 FOR i=1 TO v: LET v(i)=0: L
ET r(i)=0: NEXT i
  40 PRINT AT 16,0;"Please name
these variables"
  50 FOR i=1 TO v: INPUT "Variab
le Name?";v$(i): NEXT i
  60 PRINT AT 16,0;"Please name
the outcomes                  "
  70 INPUT "Outcome 1? ";q$
  80 INPUT "Outcome 2? ";r$
  90 CLS : FOR i=1 TO v: LET v(i
)=0
 100 PRINT "Variable: ";v$(i)
 110 INPUT "Is this variable the
case?";a$
 120 IF a$="y" THEN LET v(i)=1
 130 NEXT i
 140 LET d=0
 150 FOR i=1 TO v
 160 LET d=d+v(i)*r(i)
 170 NEXT i
 180 IF d>=0 THEN : PRINT "Outco
me is ";q$: INPUT "Is this right
?";a$: IF a$="y" THEN GO TO 220
 190 IF d<0 THEN : PRINT "Outcom
e is ";r$: INPUT "Is this right?
";a$: IF a$="y" THEN GO TO 220
 200 IF d>=0 AND a$="n" THEN : F
OR i=1 TO v: LET r(i)=r(i)-v(i):
 NEXT i
 210 IF d<0 AND a$="n" THEN : FO
R i=1 TO v: LET r(i)=r(i)+v(i):
NEXT i
 220 CLS : PRINT AT 16,0;"Any ke
y to continue"'" BREAK to stop"
 230 LET a$=INKEY$: IF a$="" THE
N GO TO 230
 240 GO TO 90
```

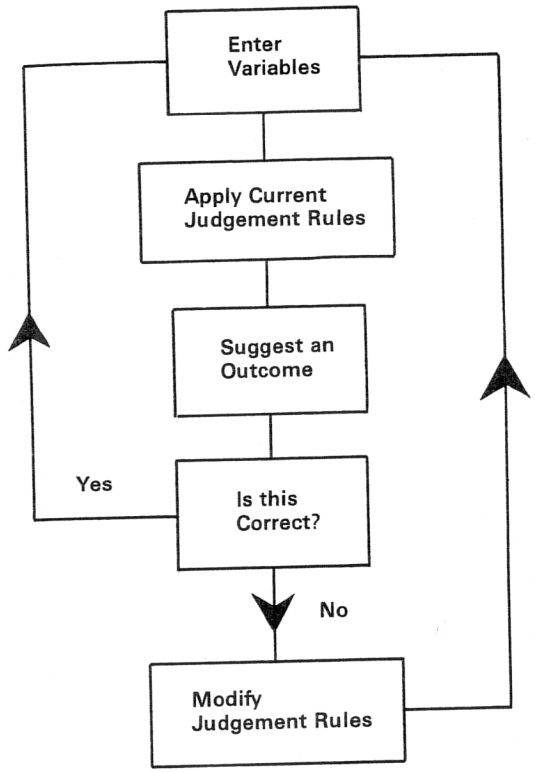



Fig 3.2
A learning process for developing a set of judgement rules.

If you enter this program and run it, what it appears to do is this:

It asks how many variables you have and DIMensions two arrays. One is to hold a judgement rule which it will develop all by itself. The other is to hold information on the variables present in a specific case. It asks for the names of the variables so that it can talk to you sensibly about them. It asks for the names of the two possible outcomes, Q1$ and Q2$. It then goes into a loop in which it asks you for the details of specific instances (is this variable the case - yes or no). Having collected details of one case it makes

a guess about what it should be and suggests an outcome. If you agree with the judgement you reply Y for YES and it goes on to another case. If you disagree it modifies its judgement rule slightly and then goes on to another case.

What you should notice is that, gradually, the machine gets better at guessing the correct outcome although how much better it gets depends rather on what you give it to work on.

If, for instance, you're still worrying about the weather and the outcomes are Rain/No Rain then it's hard to see how you could tell if the machine was right or not - unless you're willing to wait until tomorrow to find out! And one entry a day isn't very fast as far as learning speed goes.

A better scheme would be to work from a collection of weather records so that you actually knew what the outcome should be.

On the other hand, you don't have to have that kind of problem. You could invent something rather stricter - like classifying objects, for instance. Suppose you think of an object - it has to belong to one of two classes. Think of a number of variables which could be associated with it and enter those. The program has to be able to learn which class of object you're thinking of.

You may, for instance, want the 'expert' to tell you whether it's a bird or a plane (a typical, workaday problem). Your variable list then might contain items such as: wings, tail, beak, engine, feathers, undercarriage, and so on. Now, obviously, as soon as you've specified that it's got an undercarriage it's pretty likely it's a plane rather than a bird and, as you've got to specify all of the variables every time, there's no way of avoiding the undercarriage question. So the answer is quite certain and the task is to see if the expert system can be as certain about it as you are.

If we choose a bird or a plane and run the program, we find that the first time, thinking of a plane, the system guesses that it's a bird. We tell it that this is wrong. The next time it gets it right. The first time we think of a bird it gets it wrong. We tell it it's wrong. The next time it gets it right. After that it never makes a single mistake. It's become expert in judging between a bird and a plane.

Which is fine. It makes us feel as if we're getting somewhere at last. But, before we go out and celebrate, it's worth spending some time checking out just what happened and why it happened. Because what we really want to know is whether or not this program is a universal solution to all of our expert system problems.

First, we outline the variables which are set to "1" whenever they are true:

	Yes?
Wings	1
Tail	1
Beak	1
Engine	0
Feathers	1
Undercarriage	0

In other words, the variable array V, is built up of noughts and ones according as to whether or not the current variable (object under examination, call it what you like) has that particular property or not.

In the case of a bird, V is the array (1,1,1,0,1,0).

In the case of a plane, V is (1,1,0,1,0,1).

Now look into the program and find array R which is the rule the expert system has developed for judging between the two possibilities.

By the time the system has stopped making mistakes, we find that the rule array R is (0,0,1,-1,1,-1).

We form the variable D by multiplying arrays R and V so that:

D = D + R(I)*V(I) for all the values of I (i.e. 1 to 6).

So, if we take V for a bird, we get D = 0 + 0 + 1 + 0 + 1 + 0 = 2.

Taking V for a plane we get D = 0 + 0 + 0 - 1 + 0 - 1 = -2.

So the expert can say it's a bird if D is positive (greater than nought) and a plane if D is negative (less than nought).

As this is all we can say about birds and planes with these variables it's obvious that the expert system has learnt correctly and can't make a mistake any more.

Looking at the rule array again we see that there's a nought on the first two variables, wings and tail, so these aren't taken into account in making a decision. D starts off with the value 0 and the variables do nothing to

change that fact. Which is pretty reasonable because if it's got wings and a tail it could be either a bird or a plane. That knowledge doesn't tell us anything.

On the other hand, beak and feathers each have the value +1 and engine and undercarriage each have the value -1.

So it seems that the behaviour of the system looks reasonable, at least in this example.

But, at this stage, it might look as if we've done something very different from when we talked of probabilities. Have we ?

It's fairly easy to check. We can write out the table again and fill in the conditional probabilities as before:

	Bird		Plane
P(Bird:Wings)	.5	P(Plane:Wings)	.5
P(Bird:Tail)	.5	P(Plane:Tail)	.5
P(Bird:Beak)	1	P(Plane:Beak)	0
P(Bird:Engine)	0	P(Plane:Engine)	1
P(Bird:Feathers)	1	P(Plane:Feathers)	0
P(Bird:Undercarriage)	0	P(Plane:Undercarriage)	1

If it's got wings or a tail the table tells us nothing about whether or not it's a bird or a plane - the odds are equal both ways - but it has to be one or the other. The other four variables tell us that it's either a bird or a plane, but not both, and the probabilities associated with that are 1 and 0 to denote certain events. In terms of a diagram, take a look at Figure 3.3.

In this example, if it's got a beak then it must have feathers and it can't have engine or undercarriage. So it's a bird with probability 1. If it's got an engine then it must have an undercarriage and it can't have a beak or feathers. So it must be a plane with probability 1. There is no possibility of making a mistake once we know these variables.

And that is just what our expert system has done - i.e. it has developed a rule so that there is no possibility of making a mistake. Which is in line with events which occur with probability 1.

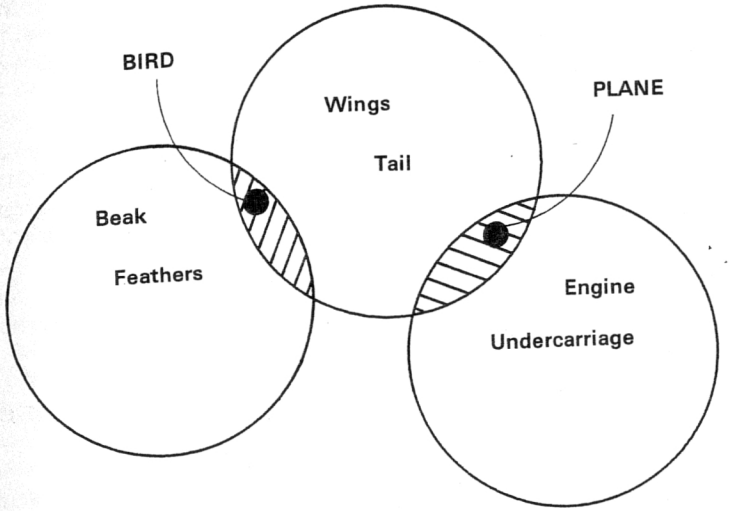


Fig. 3.3
A bird has wings, tail, beak and feathers. A plane has wings, tail, engine and undercarriage.

3.3 Other types of data

In the previous section we concentrated on cases in which the expert system worked on a simple yes/no dichotomy. It just wanted to know if a given feature was present or absent and scored either 1 or 0 accordingly.

But note: it used 1 and 0 - not the words Yes and No. So, as it's using numbers, could it use any numbers? The answer is Yes. It would make no difference at all to its working.

Which is good news. Consider the weather again. When we started off, we simply considered whether or not it was cold, wet, windy or foggy.

Now, if you glance at a weather report for yesterday you'll see that you could have had the temperature (maximum and minimum) measured in degrees, the rainfall measured in millimetres, the average wind speed measured in miles per hour, and the visibility measured in miles (well,

millimetres in England). All of these items would surely give us much more information about the weather than simple yes/no indicators. We would instinctively tend to feel that we were losing something if we disregarded all of this potentially useful data.

So we don't disregard it. Instead of having the expert ask if a given variable has occurred we could have it print the name of each variable and take in a numeric value by way of response. So we could give the temperature in degrees, the rainfall in millimetres, the visibility in miles, and so on.

And, if we wanted to include a simple yes/no item - such as Thunderstorms Yes/No - we can continue to do so by replying, say 1 for the presence of that feature and 0 for its absence.

The program will then proceed as before but might, hopefully, give better results because we're giving it better information.

However, to make it clearer as to what's happening most of the examples that follow will use discrete, dichotomous (Yes/No), data - but remember, you could have continuous variables just as easily, and in some cases they might be better.

On the other hand, we can't do this with the outcomes.

It can only choose between different categories of outcome. It can't produce an answer which says exactly how much rainfall it thinks there will be tomorrow - it can only say whether or not it thinks there will be rainfall.

This doesn't mean that we can't make a rough attempt at getting it to be more specific. We could decide that there will be rainfall falling, as it were, into five categories:

Rainfall Categories:

No Rain	Less than .5"	.5" to 1"	1" to 1.5"	Over 1.5"
1	2	3	4	5
Dry	Wet	Wetter	Really Wet	Horrid

In this case our expert system could, with five outcomes, say something a bit more sensible about the likely weather. And shortly we'll be seeing how to devise a system with this number of outcomes.

This doesn't mean that a system could not be devised to try to give exact numeric answers rather than simple categories. If we wanted to we could try, say, developing a multiple regression equation for rainfall based on a large number of input variables. This would give an exact estimate (exact estimates are not necessarily correct predictions) of tomorrow's rainfall.

The problem is that, although this system might work for predicting rainfall, it couldn't easily be modified to be expert in any other field. How, for instance, would it work out if that was a Bird or a Plane? (Or, for that matter, a Glider?)

The object of building your own expert system is to produce something which can become reasonably expert in a number of different fields. A sort of general-purpose device which you could apply to something that interests you. And, that being the case, there's no point in enabling you to predict the weather if what you really want to do is diagnose your own medical symptoms so that you can be a successful brain surgeon, or some such.

3.4 The judgement rule

Maybe you've now set up the learning system and found that it does seem to, more or less, work. Fine, you say. But, exactly, why does it work? What's it doing and can it be made to do it any better?

Well, consider the idea of Description Space for, not only is it a useful thing to consider, it's also pretty impressive to be able to talk knowledgeably about other kinds of space than the boring old three-dimensional kind that you see before you.

To make it easy though, we'll start off by assuming that you're sitting at your desk which, conveniently, has a two dimensional top spread out before you in normal common-or-garden space.

Also, you have two sets of objects - pencils and paperclips say - which we can use to illustrate some points.

Now suppose that you want to get your heap of pencils and paperclips and divide them into two heaps - one containing only pencils and the other containing only paperclips. This is easy. Even the simplest amongst our midst can achieve this task - all you have to do is to take each object in turn, look at it, say 'Is this a pencil or a paperclip?' and place it in one or other of the heaps according to the answer you give yourself. It is as easy in fact as looking at an object and wondering if it's a bird or a plane - and

categorising it depending on which it is. The point is that the problem of classifying the object is trivial if the description of that object coincides neatly with the criteria by which we want to carry out the classification.

But suppose that, for some reason, when you looked at the object you couldn't immediately tell if it was a bird or a plane, or a pencil or a paperclip? Then the problem is definitely non-trivial and you have to think of some other method of passing judgement. And this is exactly the position that the computer is in - it can't see the big difference, even if it's obvious to you, so it has to get a little more subtle.

So, now get the pencils and the paperclips and spread them out on the desk. Place the pencils on the left and the paperclips on the right. Now, ignoring the fact that you can identify pencils and paperclips at a glance, how would you decide which was which? Well, it's easy. All you have to do is to draw a line down the middle of the desk so that all of the pencils are on the left of the line and all of the paperclips are on the right. Now you can classify the two sets of objects simply by measuring their position along the desk.

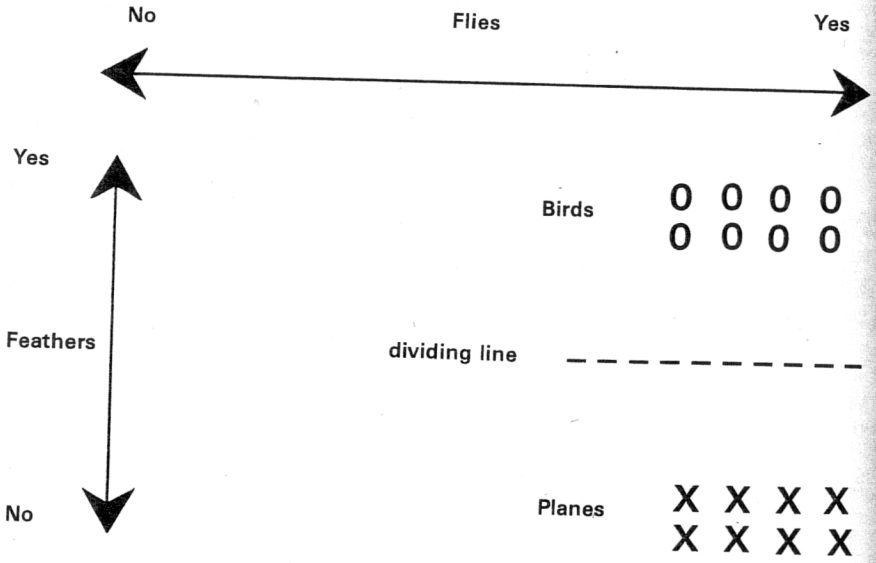


Fig 3.5:

A Description Space that Consists of Two Dimensions specified as 'Flies' and 'Feathers'. Planes can fly but have no feathers. Birds can fly but have feathers.

And that, roughly, is what the computer is doing. Except that it's got to work a little harder than you because the objects haven't been laid out neatly in real space. They've been laid out in a particular description space - which isn't really very different.

Go back to the bird/plane example now and suppose that two items of information are available. Does the object fly? And, does the object have feathers?

Now clear your desk and write 'flies' along the length of the desk top and 'feathers' along the depth of the desk top as in fig. 3.5. If an object can fly then it's placed to the right of the desk, and to the left if it can't. If an object has feathers it's placed at the back of the desk, and to the front if it can't.

Obviously, if you had a bird in your hand (!) you would place it at the back and to the right. If it were a plane, then you'd place it at the front and to the right. And, if you wanted to know whether an object was a bird or a plane - without being specifically told what it was - you could answer accurately every time simply by looking to see which quarter of the desk it was on. You could make a judgement simply by drawing a line between the two heaps of objects and seeing which side of the line any object fell.

And it wouldn't have to be simply Yes/No items such as feathers/no feathers. You could, if you wanted, consider a bit of weather forecasting and put a scale along the desk top to denote rainfall figures, say.

If you throw away the desk, you simply have any number and type of objects hanging there in description space. It isn't a space defined by the desk anymore. And, really, it isn't anything to do with normal three dimensional space anymore. It's a brand new type of space which is defined solely by the variables you're using - one axis to each variable. And the values on these variables that any object posseses uniquely define its position in this description space. Unfortunately, we can't draw a picture for you!

Just to illustrate the idea in three dimensions, suppose you were out walking in the garden one fine summer's evening (we do get fine summer evenings occasionally). Suppose that you see two clouds of midges buzzing around in the sunlight. One cloud of midges is made up of Big Midges and the other of Little Midges and you decide, perversely, to try to separate the two. One way which might work would be to get the biggest sheet of paper in the world and place it between the two clouds of midges.

This might work but, before you try it, the only reason for giving this example is to show that you can have two groups of objects which are

defined by reference to three dimensions (the physical position of each midge) and that you could, in theory, separate them by placing a surface between them.

In fact, once you've moved into description space, the objects can be defined in terms of any number of dimensions. If you have one dimension for each measured variable then a set of objects defined in terms of ten variables exist in a ten-dimensional description space. But the idea about placing a surface between them is just the same - you can have a ten dimensional surface for your ten variables and each object can be classified according to which side of the surface it falls.

Incidentally, the equation of a surface in n dimensions is $y = b_1x_1 + b_2x_2 + ...+ b_nx_n$, where the x_i represent 'position' measurements on the n axes and the b_i are constants.

So the desk top would have the equation, in two dimensions, $y = b_1x_1 + b_2x_2$

By specifiying the x_i for a new object we can calculate y for that object and we could use the value of y to say on which half of the surface the object occurred.

And that is what the learning program does. It takes all of the x_i we can give it by way of examples and calculates some values for the b_i so that, once it's working properly, it gets a value of y which uniquely determines which group an object should belong to.

The way you can think of it is as a process of trial and error. You present the program with an object and values for its variables. Initially, the program has a surface which is lying around just anywhere and it looks to see what side of the surface your object falls and decides accordingly. It may, by chance, be right. But if it's wrong, it nudges the position of the surface around a bit in description space using the values you gave it. It then has another go with a new object, maybe doing a bit more nudging of the surface. Then again, a bit more nudging, again... And so on, until it has the surface positioned so that it will always come up with the right answers - if, in fact, it's possible to do so.

Having given this explanation you might well wonder if there's a 'best' way of using this particular method.

Would it, for instance, be better to adjust the position of the surface in some way when the computer guessed right as well as when it guessed wrong?

The simple answer is that it all depends on the type of problem you give it - but not in a simple way. If you have some data to present to your expert system then the following points are worth bearing in mind:

If there's only a finite number of possible examples that can occur you'll get the best results by giving the expert all of these possible values to learn from.

It may take a while to get the surface (the judgement rule) into the correct position - one showing of each example often won't do. So you may need to present all of the examples a number of times.

It's difficult to say beforehand exactly how many goes the system will need to get things right - so you should work pragmatically (which means you suck it and see). Give the expert a set of examples and let it have one go at them and count how many mistakes it makes. Then let it have another go and count the mistakes. Carry on like this until either it's not making any mistakes any more or until it's not improving any more.

If you want it to work on examples which you do not, as yet, have (like, for instance, in predicting the weather at some future date) then try to make sure that the examples you give it to learn from are as much like the ones it will eventually get as is possible. This may sound obvious - but if you train it on one thing it may not necessarily be able to use the same rules on other, rather different, data.

There is one point you might be wondering about - if expert systems are to replace human experts by embodying human expertise, how is it that the system worked out a set of rules all by itself? It seems almost as if it didn't need any human help at all. The answer is that you did, maybe without realising it, give it quite a lot of very human knowledge. What you gave it was the initial knowledge of which object was which - it was only because you, the user, were able to judge the difference between a bird and a plane that enabled the program to learn at all.

3.5 Building a rule

Once you've realised that all the learning system does is to find the equation of a surface in n-dimensional description space, such that this surface reasonably well separates two groups of objects, you might well think that it's possible to do something rather better.

After all, the learning system finds its surface in an apparently rough and ready way and needs, often, fairly lengthy training. Why not, therefore,

calculate a really good set of values for a surface and give those values to the expert system from square one? That way there's no training needed and, in cases where it's rather hard to judge between objects, you would at least know that the best possible surface was being used.

Well, without wanting to put a damper on any enthusiasm it's only fair to point out that the subject of discrimination is a rather hard one. Incidentally, some people refer to the subject as 'discrimination' others as 'classification' but they mean the same thing. And, typically, in books on multivariate analysis (multivariate = lots of variables) you'll find a chapter or two on the problem.

Consider again the problem of the two clouds of midges which you want to separate - this is fine as long as the two clouds are reasonably far apart, but if they aren't not only is the learning system likely to make mistakes but almost any system will make mistakes because there may not be any position in which a surface could be placed to separate the two groups.

But what could be done is to calculate the exact centre of each group, draw a line between the two centres, and then classify each individual item according to which side of the line's centre point it fell. This, in fact, is the basis of most methods of discrimination. In many ways it's better to do this than to rely on the learning system because the learning system relies heavily on outlying values when deciding the final position of its surface. That is to say, it only alters its surface when it makes a mistake and, towards the end of the process, it only makes mistakes when it's nearly in the right position anyway. So, if you think of the two clouds of midges, the surface is being moved around at the whim of a very few midges right on the edge of the cloud - and these might not be very representative of the positions of the midges as a whole. And it's the same with the objects in description space. Outlying objects may not be representative and might give some unwanted results.

The situation that we have is a problem of classifying objects in an N-dimensional space. So, we start by reserving space,

```
10    INPUT "M1 = ";M1: INPUT "M2 = ";M2: INPUT "N = ";N
20    DIM V1(N),V2(N)
```

for the N variables and then find M examples. Suppose we have M1 of the first kind and M2 of the second kind.

Then, we calculate the average value of each variable, V1(J):


```
30    FOR I = 1 TO N
40    V1(I) = 0
50    NEXT
60    FOR I = 1 TO M1
70    FOR J = 1 TO N
80    INPUT "VARIABLE = ";X
90    V1(J) = V1(J) + X / M1
100   NEXT : NEXT
```

where X is the value of variable J for example I

Do the same for V2(J) on the M2 examples of the second kind and you then have the average values or "means" for each of the two kinds in the two arrays V1 and V2.

```
110   FOR I = 1 TO N
120   V2(I) = 0
130   NEXT
140   FOR I = 1 TO M2
150   FOR J = 1 TO N
160   INPUT "VARIABLE = ";X
170   V2(J) = V2(J) + X / M2
180   NEXT : NEXT
```

Now, try to classify another object using these means. Suppose that you have the values of the variables of this object in array X. Don't forget to DIM X(N).

The 'distance' of the new object from the two means can now be calculated as D1 and D2:-

```
190   DIM X(N)
200   D1 = 0:D2 = 0
210   FOR J = 1 TO N
220   INPUT X(J)
230   D1 = D1 + X(J) * V1(J)
240   D2 = D2 + X(J) * V2(J)
250   NEXT
```

Alternatively, you could calculate the differences between the means and store them in a new array V.

```
260   DIM V(N)
270   FOR J = 1 TO N:V(J) = 0: NEXT
280   FOR J = 1 TO N
290   V(J) = V1(J) - V2(J)
300   NEXT
```

Then, you could simply calculate D from array V (DIM'd as V(N)) and array X and classify the object depending on whether or not D was greater than or less than zero.

```
310   D = 0
320   FOR I = 1 TO N
330   D = D + X(I) * V(I)
340   NEXT
```

At which point it becomes fairly plain that this is, in terms of code, very similiar to the method used by the learning system.

In fact, it's possible to modify the learning system so that it turns into this 'nearest mean' method.

Instead of adjusting the decision rule only when a mistake is made, give the system an example and tell it which class the example belongs to. Then alter the decision rule so that it constantly holds the latest value for the difference between the two sets of mean values.

For instance, if there have been M1 examples from class one so far and you then provide another class one example,

```
370   FOR J = 1 TO N
380   V1(J) = (V1(J) * M1 + X(J)) / (M1 + 1)
390   NEXT
```

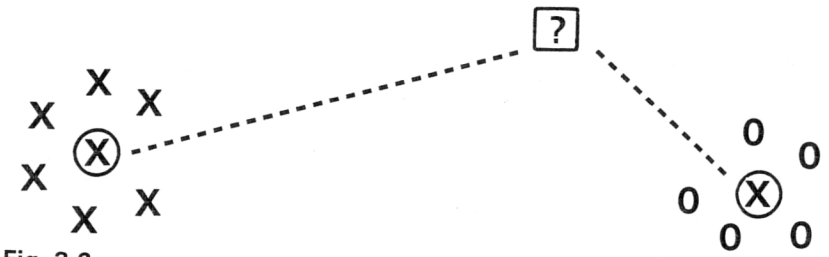


Fig. 3.6
Measuring the distance to the nearest mean

where the array X holds the values for the new object.

This way V1 is constantly updated with the latest estimates of the mean value for that group.

And, using the latest values in V1 and V2 it's possible to have the latest values for the decision rule, say

```
FOR J=1 TO N:R(J) = 0:NEXT
FOR J=1 TO N
R(J) =V1(J) - V2(J)
NEXT
```

and then to make subsequent decisions according as to whether D is greater than or less than zero with:

```
D=0
FOR J=1 TO N
D=D +X(J)*R(J)
NEXT
```

There are plenty of other ways of constructing decision rules which could be used but they all depend on a more intricate knowledge of the variables being used. In the meantime, if you want to know which of these two methods is best - the learning algorithm or the nearest mean approach - the best way is to try it and see. There isn't a general answer because so much depends on the exact nature of the variables which you, personally, choose to give the system.

3.6 Prior probabilities

But there is one refinement to the nearest mean approach which you might try. And this occurs in the case where you know for sure that one outcome is more likely than another, irrespective of the values of the variables. This is known as the prior probability of each outcome and if outcome one was three times more likely to occur than outcome two, then P(outcome one) = .75 and P(outcome two) = .25. What you could do in this case is to make a decision which wasn't quite based on the nearest mean.

Consider: to make a decision based on the nearest mean with the array R holding the difference between the two means you would simply consider whether the calculated D was greater than or less than zero. But maybe, if D was calculated as approximately zero for some example and you knew that class one, say, was much more likely than class two then you might decide to go for class one even if the calculated value of D tended to suggest the opposite.

In programming terms it simply amounts to testing for D greater than some value C (say) and, whereas, normally C=0 it might improve matters a bit if C had some other value.

The difficulty lies in specifying a really good value for C other than zero. If you know enough about the data you're working with and are a competent mathematician it's possible to find a value which enables you to say some fairly definite things about how the system will behave. But if you don't and you aren't, then it's back to experimenting (which isn't a bad idea anyway) and the only real warning that needs to be given is that if you don't have enough typical examples to test the system on then you'll never really know how it's likely to work in practice at all and it would be better to avoid any complications.

3.7 Expanding your options

So, there you are, staring up at the sky. All around you people are crying out: "Is it a bird? Is it a Plane?" You reach for your expert system and key in a few well-chosen variables and, in seconds, you make an authoritative pronouncement. "Wrong!" the people exclaim in tones of malicious glee. "It's a glider."

It makes you feel every bit as bad as do your weather forecasts. Turning to your finely-wrought expert system it predicts Rain for you. Or, maybe, it predicts a dry day. But either way, as you grope your way around in an impenetrable fog the following day you feel kind of let down. Fog, you suppose, is kind of wet. But not completely wet like rain. And it's also kind of dry. What fog really is, you muse, is Fog. And nothing else.

And, somehow, you feel cheated that your expert system should have failed to allow for other outcomes.

In general, you want your expert system to be able to give any number of different outcomes. You might rest content with having to specify these outcomes in advance - but you still want more outcomes than just two, which is all we've had so far.

There are, of course, lots of ways of building an expert system so that you can have a variety of outcomes. But, for the time being, we'll just consider an extension of the system we've dealt with so far.

Recall that we had a rule for making decisions. It was one rule and the expert system made a decision - effectively, a yes or no decision - on the basis of that one rule. What we'll do now is provide the system with a general number of rules so that it can pronounce on a general number of possible outcomes. To do this we draw up a two-dimensional array to keep the rules in, and this array will look very much like the rectangular matrix in which we first put our probabilities.

Call the array R(V,K) - the rules for V variables, with K outcomes.

And we will use the bird/plane example again:

	Bird	Plane
Wings	0	0
Tail	0	0
Beak	0	0
Engine	0	0
Feathers	0	0
Undercarriage	0	0

For the moment, we'll forget about gliders and, with 6 variables, we have R(6,2), so K=2 outcomes.

The method of working is this:

Each column of the array R contains a rule which is used to indicate how strongly the expert system believes in the outcome corresponding to that column. The strength of the belief is measured by calculating D, as before, using array V which contains the values of the variables being considered at the moment.

So: for outcome J (J=1 or 2) we calculate D as:

D = D+R(I,J)*V(I) for I=1 to 6

You notice that, to start with, all R(I,J) = 0 so, obviously all D = 0. In other words both beliefs are equally strong (or weak) at this stage.

Now, depending on how you want to run things, you either tell the expert which group the array V belongs to, or you let it guess and, if it's wrong, you let it take the following action:

If V belongs to outcome Bird then we add the V(I) values into the Bird column and subtract them from the Plane column. If V belonged to Plane we would add the V(I) values to Plane and subtract them from Bird. This is what it looks like after adding the 'Bird' values:

	V(I) Bird	V(I) Plane	R(I,1) Bird	R(I,2) Plane
Wings	1	1	1	-1
Tail	1	1	1	-1
Beak	1	0	1	-1
Engine	0	1	0	0
Feathers	1	0	1	-1
Undercarriage	0	1	0	0

So far, we've presented a V for Bird to the array R. It couldn't make a decision because all of the D were identical. So it added V(I) to column one and subtracted it from column two. Now, if it gets a Bird again it can make an accurate decision because D will evaluate as +4 using R(I,1) and as -4 using R(I,2). So all the expert has to do is to select the column of rules which gives the biggest value to D.

But, if we now give the expert a V(I) for Plane we find it will still select Bird as the most likely outcome as, using R(I,1) gives D a value of +2 and, using R(I,2) gives D a value of -2.

So, our amazing expert system can't even tell the difference between a bird and a plane yet.

But add V for Plane into the second column and subtract V for Plane from the first column, and we get:

	R(I,1) Bird	R(I,2) Plane
Wings	0	0
Tail	0	0
Beak	1	-1
Engine	-1	1
Feathers	1	-1
Undercarriage	-1	1

Now, if we present it with V(I) values for Plane, column one gives D a value of -2 and column two gives D a value of +2

So, it will choose the outcome: Plane. Correctly.

And, if you re-check with V(I) for Bird you'll find that this gives +2 on column one and -2 on column two.

So the expert system now works every bit as well as it did before.

In fact, if you check back, you'll find that it's doing just the same thing as it was doing before when we only had a one-dimensional rule array - so you might, rather cynically, think that this has been a lot of effort for nothing.

And, if you just want to decide between a bird and a plane, it certainly was a lot of effort for nothing.

But, bring back that glider and add another column to R(I,J) to allow for it. You'd proceed just as we have done with more than two options. You calculate the values of D for each column and choose that column which gives the maximum value of D. In the event of a mistake, or a tie, you tell the system which column it should have chosen. The system then adds the V(I) values it's working with to the correct column and subtracts them from any other column which gave a value of D greater than, or equal to, the column it should have chosen.

If you like, it gives a lift to the correct outcome's rule and pushes down the values of the other outcomes' rules to get them out of the way of the correct one.

Suppose we'd had a glider. A glider, of course, is like a plane but it doesn't have an engine or undercarriage.

Now, on the first guess, given a Bird, all the R(I,J) = 0 so it might have chosen Glider as well as the other two options. So the third, Glider, column is pushed down in the same way as the Plane column.

On the second try we gave it a Plane - and it guessed a Bird. So we added V(I) for Plane to the second column and subtracted V(I) for Plane from the first column. But now that we have a third, Glider, column which so far is equal to the plane column we also have to subtract the V(I) from the Glider column. This gives:

	R(I,1) Bird	R(I,2) Plane	RI,3) Glider
Wings	0	0	-2
Tail	0	0	-2
Beak	1	-1	-1
Engine	-1	1	-1
Feathers	1	-1	-1
Undercarriage	-1	1	-1

Now,obviously, like this it's never going to select Glider because with every value in column three being negative there will always be one of the other rules which gives a bigger value of D. But give it a Glider anyway.

	V(I) Glider	R(I,3) Glider
Wings	1	-1
Tail	1	-1
Beak	0	-1
Engine	0	-1
Feathers	0	-1
Undercarriage	0	-1

We don't need to show the other two anymore because everytime a Glider turns up, and the expert fails to guess what it is, the same values are subtracted from both Bird and Plane - minus one from the wings and tail rules. This won't affect the system's ability to guess between a bird and a plane at all. But, gradually, adding +1 to wings and tail in Glider will pull up the values there until when a Glider arrives R(I,3) will give the maximum value of D and a correct choice will be made every time.

By that time R(I,J) will look like this:

	R(I,1) Bird	R(I,2) Plane	R(I,3) Glider
Wings	-1	-1	0
Tail	-1	-1	0
Beak	1	-1	-1
Engine	-1	1	-1
Feathers	1	-1	-1
Undercarriage	-1	1	-1

You can check for yourself that this will now choose correctly between the three alternatives. It's learned correctly and it won't make any more mistakes.

And, what's more, you could have had *millions* of different outcomes and *millions* of different variables and it would still have kept on churning the numbers around until if found something like a correct set of rules for choosing an outcome from a particular set of variables.

3.8 Can it make a mistake?

So far so good. But what are the odds on the thing going wrong - or never learning the correct outcomes adequately?

Well, the odds are pretty difficult to give in terms of exact probabilities because it depends so much on what type of problem you set the expert.

The method all depends on the concept of Linear Separability and this is a good phrase to remember.

"Of course," you pronounce as you lean casually against the bar of your choosing, "it works perfectly, does my Expert System, once it's built up its Knowledge Base." At this point you might try giving a thoughtful suck at your pipe, for effect, and add: "As long as it's working on a problem that's Linearly Separable, of course." At this point you might try a deprecating laugh and exclaim: "But show me a problem that isn't!"

This show of jargon should daunt all but your severest critics and will easily make up for the fact that your expert system doesn't always, as it were, work.

It's simply back to our clouds of midges. As long as you can draw a line or place a surface between the two groups this expert system will learn to distinguish between them. If you can't, it won't. It's as simple as that.

Consider, though, the example of the red and blue garden railings.

You have a garden with a line of railings down the side of it. They alternate: a red one, a blue one, a red one, a blue one (you'll soon get the hang of it). Are these railings linearly separable? Can you distinguish between the red ones and the blue ones by placing a surface between them?

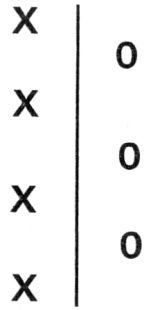
Well, it depends. If you measure them (or describe them) in terms of distance down the garden, you can't. Suppose you did so describe them. You notice that ten feet down the garden is a red railing. Twelve feet down the garden is the next railing and it's blue. So you place a surface eleven feet down the garden between the two railings. Obviously, it distinguishes between those two railings - one red and one blue - but it fails to distinguish between all of the other railings both this side of the surface and the other side.

Alternating railings aren't always linearly separable.

X
0
X

0
X
0
X

It's impossible to draw a line that separates one type of railing from another.....



...unless we can describe them in a different way that makes use of their differences.

Fig. 3.7
The alternating railing problem

The railings, as you've described them, are not linearly separable.

If, however, you went down the garden and moved each red railing a little to the left and each blue railing a little to the right then you could easily place a surface between them simply by placing your surface along the original line of the railings. Then, they'd still be the same railings but they would, now, be linearly separable.

Alternatively, you could stop describing them in terms of where they were in the garden and simply number them in order. As they alternate, first one and then the other,it's obvious that the even-numbered railings will be one colour and the odd-numbered railings another colour. So, if you described all of the railings in this way they would be linearly separable and our expert system could learn to tell which was a red railing and which was a blue railing.

The point is: that the railings haven't changed. You haven't even moved them about the garden. You've simply described them differently.

That's the essence of the matter. You must try to choose variables which look as if they're going to enable the expert system to distinguish between the various outcomes. Not a very subtle point to make, to be honest, because you'll naturally tend to do this anyway. After all, if you want your expert system to say whether or not it's going to rain tomorrow you're hardly likely to give it, say, the football results to work on. Given a knowledge of today's football results won't help it predict tomorrow's weather so you aren't making the problem linearly separable by doing so.

(One might concede that the football results say something about today's weather, from which something might be deduced about tomorrow's weather, but one would hardly expect a clear-cut answer.)

Apart from the nature of the problem itself (are the cases linearly separable?) there is the matter of how long the expert system will take to learn a decent set of rules.

The best way is to give it all possible examples until it stops making mistakes if the number of possibilities is fairly small. If there's an enormous number of possibilities then you at least need to give it a couple of examples for each possible outcome and keep an eye on its operation until it seems to have settled in a bit.

The program works by using the methods we've described so far - but it's worth running through it just to make sure that nothing's wrong.

First, it wants to know how many variables are to be considered in the decision-making process and how many outcomes there can be.

These are the values V and Q. Knowing these values the arrays can be DIMensioned. We have: V(V) for holding the values of the current variables; V$(V) for holding the variable names; Q$(Q) for holding the names of the outcomes; R(V,Q) for holding the decision rules of the V variables for the Q possible outcomes; D(Q) for holding the values calculated for a given V(I) using R(I,J) for the Q possible outcomes.

The names of the variables and outcomes are entered so that they can be referred to by name.

At line 120 a training session begins. Values for V(I) are entered and, using R(I,J), the Q values of D(J) are calculated.

Array D is searched for its largest value and a guess is made that this largest D(J) = D(HI) points to the correct outcome, Q$(HI).

If this is the correct outcome the program returns to line 120 to continue training with another example.

If it isn't, all of the possible outcomes are displayed and you are asked to say which was the correct outcome. This is Q$(HJ).

Using this knowledge the system readjusts R(I,J) accordingly, subtracting V(I) from each rule that gave a value as big as or bigger than D(HJ) and, finally, adding V(I) to the rule HJ i.e. R(I,HJ).

Having adjusted R(I,J) the program then returns again to line 120 for another example.

As the training session proceeds the judgement of the system should improve and, in some cases as the bird/plane/glider example, will eventually get it right every time.

And there's a bit of inbuilt scoring so that the program can judge when it's perfect for itself.

Fig. 3.9
The program so far.

Apple II listing

```
10    HOME : INPUT "HOW MANY VARIABLES HAVE YOU ?";V
20    DIM V(V),V$(V)
30    PRINT "PLEASE NAME THESE VARIABLES"
40    FOR I = 1 TO V
50    PRINT "VARIABLE ";I;" IS ";: INPUT V$(I)
60    NEXT
70    INPUT "HOW MANY OUTCOMES HAVE YOU ?";Q: DIM Q$(Q),R(V,Q),D(Q),S(Q):
      REM :WITH S(Q) WE CAN KEEP A SCORE
80    PRINT "PLEASE NAME THESE OUTCOMES"
90    FOR I = 1 TO Q:S(I) = 0
100   PRINT "OUTCOME ";I;" IS ";: INPUT Q$(I)
110   NEXT
120   HOME : PRINT "THIS IS A TRAINING SESSION": PRINT "PROVIDE VALUES OF
      VARIABLES": PRINT "I WILL GUESS AN OUTCOME": PRINT "YOU MUST TELL ME IF I
      AM RIGHT OR WRONG"
130   D = 0: FOR I = 1 TO Q:D(I) = 0: NEXT
140   FOR I = 1 TO V
150   PRINT "VARIABLE ";I;" (";V$(I);") IS ";: INPUT V(I)
160   NEXT
170   FOR I = 1 TO V
180   FOR J = 1 TO Q
190   D(J) = D(J) + V(I) * R(I,J)
200   NEXT : NEXT
210   FOR I = 1 TO Q
220   IF D(I) > = D THEN :D = D(I):HI = I
230   NEXT
240   PRINT "IS IT OUTCOME ";HI;" (";Q$(HI);") ?"
250   INPUT A$
260   IF A$ = "Y" THEN :S(HI) = 1:SC = 0: FOR I = 1 TO Q:SC = SC + S(I): NEXT : IF SC = Q
      THEN : PRINT "I'M PERFECT !": END
270   IF A$ = "Y" THEN : GOTO 120
280   FOR I = 1 TO Q: PRINT I;" ";Q$(I)
290   NEXT
300   INPUT "WHICH OUTCOME IS IT ?";HJ
310   FOR I = 1 TO Q
320   IF D(I) > = D AND I < > HJ THEN : FOR J = 1 TO V:R(J,I) = R(J,I) - V(J): NEXT
```



```
330   NEXT
340   FOR J = 1 TO V
350   R(J,HJ) = R(J,HJ) + V(J)
360   NEXT
370   PRINT "I GOT ";SC;" RIGHT BEFORE I MADE A MISTAKE !": PRINT : PRINT "PRESS ANY
      KEY TO CONTINUE": GET X$:
380   FOR I= TO Q : S(I) = 0 : NEXT
390   GOTO 120
```

Sinclair Spectrum listing

```
  10 CLS
  20 INPUT "How many variables?"
;v
  25 DIM v(v): DIM v$(v,20)
  30 PRINT AT 16,0;"Please name
them:"
  40 FOR i=1 TO v: INPUT "Variab
le ";(i);" is ";v$(i): NEXT i
  50 CLS
  70 INPUT "How many outcomes ha
ve you?";q
  75 DIM q$(q,20): DIM r(v,q): D
IM d(q): DIM s(q)
  80 PRINT AT 16,0;"Please name
them"
  90 FOR i=1 TO q: LET s(i)=0: I
NPUT "Outcome ";(i);" is ";q$(i)
: NEXT i
 100 CLS
 110 LET sc=0
 120 PRINT "This is a training s
ession": PRINT : PRINT "Please p
rovide values of ": PRINT "varia
bles": PRINT : PRINT "I will gue
ss an outcome": PRINT : PRINT "T
ell me if I am correct"
 125 PRINT : PRINT "Press a key
to continue"
 126 LET a$=INKEY$: IF a$="" THE
N GO TO 126
 128 CLS : PRINT AT 16,0;"Enter
the variables:"
 130 LET d=0: FOR i=1 TO q: LET
d(i)=0: NEXT i
 140 FOR i=1 TO v
 150 INPUT "Number ";(i);" (";(v
$(i));") is ";v(i)
 160 NEXT i
 170 FOR i=1 TO v: FOR j=1 TO q:
LET d(j)=d(j)+v(i)*r(i,j): NEXT
j: NEXT i
```

```
 210 FOR i=1 TO q
 220 IF d(i)>=d THEN LET d=d(i):
 LET hi=i
 230 NEXT i
 240 CLS : INPUT "Is it outcome
";(hi);" (";(q$(hi));") ?";a$
 260 IF a$="y" THEN LET s(hi)=1:
 LET sc=0: FOR i=1 TO q: LET sc=
sc+s(i): NEXT i: IF sc=q THEN PR
INT "I'm perfect!": STOP
 265 IF a$="y" THEN GO TO 128
 270 FOR i=1 TO q: PRINT i;" ";q
$(i): NEXT i
 300 INPUT "Which outcome is it?
";hj
 310 FOR i=1 TO q
 320 IF d(i)>=d AND i<>hj THEN :
 FOR j=1 TO v: LET r(j,i)=r(j,i)
-v(j): NEXT j
 330 NEXT i
 340 FOR j=1 TO v: LET r(j,hj)=r
(j,hj)+v(j): NEXT j
 370 PRINT "I got ";(sc);" right
 before I made a mistake!": PRIN
T : PRINT : PRINT "Press any key
 to continue"
 380 LET a$=INKEY$: IF a$="" THE
N GO TO 380
 390 FOR i=1 TO q: LET s(i)=0: N
EXT i
 400 GO TO 128
```


Chapter 4

Improving your Expert

4.1 Parallel and sequential decisions

We've now reached the stage at which we actually have an expert system. After a suitable training period it should be making decisions for us on the subject of our own choosing and, unless we start examining R(I,J) we won't even need to know exactly how it's doing it.

Which is enough, really, to make most people feel pretty pleased with themselves. Apart, of course, from the inevitable sceptics.

"If it was a real expert system," they whine, "it wouldn't work like this. It would work differently." they say.

And they would point to the fact that, every time they wanted an expert decision, they'd had to key in answers to a whole load of questions all at once before the expert deigned to do a stroke of work.

Real expert systems, they argue, ask you one question, then maybe another question, and then, depending on the outcome, either ask you more questions or tell you what their expert opinion is. They don't start off by asking you absolutely everything all at once.

What, in fact, is being said is that many systems use sequential decision procedures, not parallel ones.

What we have is a parallel decision procedure - it gets in all the information on a subject and then makes a decision.

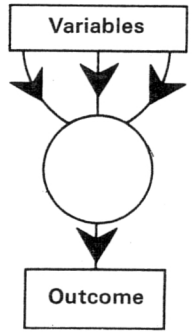
The sequential procedure is always guided by the last piece of information it received. For instance, in our weather example, a sequential procedure

might decide that the most important thing to know in passing opinion on tomorrow's weather was: is it raining today? It would ask that question and then either seek more information or pass judgement.

One of the difficulties with the weather example, though, is that we don't exactly know what will accurately predict rain tomorrow. Not so the bird/plane example - by definition we know the differences between birds and planes. So consider a sequential expert at work on this problem.

You state that you are thinking of either a bird or a plane and want the expert to say which it is.

Parallel procedures consider all of the evidence before making a decision



Sequential procedures see if they can predict an outcome after each new input. If not they request more information

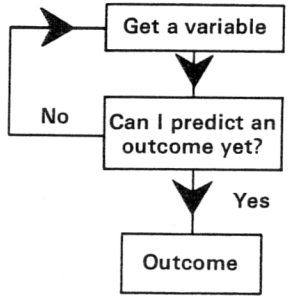


Fig. 4.1
Parallel and sequential decisions

"Does it," muses the expert as he sucks on his pipe, "have feathers?"

"Actually, yes.", you concede.

And at once the great thinker deduces that it's a bird.

There was no need to consider all six variables at all.
Suppose, now, that you were thinking of an object and didn't specify that it could fly. The parallel and sequential approaches for this problem are shown in Fig. 4.2.

"Does it have wings?" you get asked. You concede that it does and are instantly confronted with the previous question concerning feathers. At once the expert has it. The sequential technique has won.

In general, sequential techniques ask less questions than do parallel techniques - which might make them seem, initially, more desirable. Or, at least, less garrulous.

But whereas the desirability of sequential procedures is obvious in the bird/plane example it isn't in the weather example.

Because, even if you were a real expert and knew everything about the weather there is to know, you'd still never be absolutely certain if it was going to rain tomorrow or not. And, as this is the case, the wise expert would gather all possible evidence before him prior to making any judgement. And that's what parallel procedures do.

Knowing that one can't be absolutely certain, in general, about an outcome they gather all the information they can and then make the best guess they can. A sequential procedure smacks of leaping to conclusions - and our expert system doesn't do that.

If you think about it for a moment you'll realise that the best a sequential procedure can do is only as good as a parallel procedure - because, as a parallel procedure uses all of the information to come to a conclusion then the parallel procedure comes to the most likely conclusion to be right. The sequential procedure can, by sequentially going through all of the available information, come to the same conclusion but, by doing so it's effectively done just what the parallel procedure did but it did it by stages.

There isn't always much difference between the two methods. Suppose our expert had asked about the variables by clearing the screen in between each item. Then it might have looked like a sequential procedure - but, as it didn't decide anything until the end, it's really parallel.

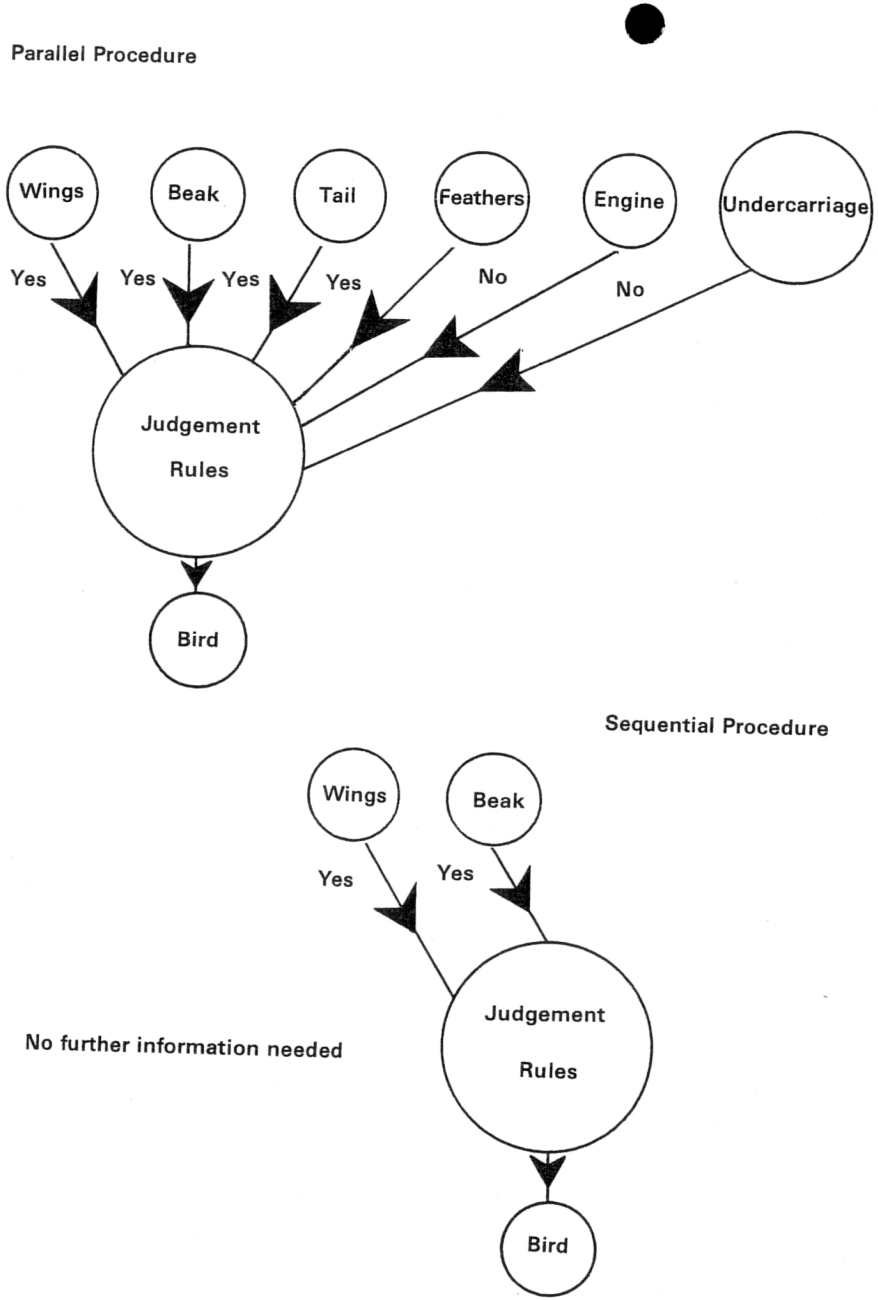


Fig. 4.1
Parallel and sequential judgements

We could obviously take some of the effort out of entering information if, when it had a crucial piece of information, the expert was instantly able to come to an opinion and skip the rest. But how would we write such a program?

Well, if we knew what the subject of our expert's enquiries was going to be we could write a tailor-made program for it. But, in our case, we wanted a general-purpose expert system.

And, if we knew the exact probabilities for every outcome we could also write a program to do it. For instance, as the expert gathered, sequentially, pieces of information, it could look through all the possibilities to find the most probable one. It could then check all other outcomes to see if they could ever, given the current information, become more probable than the one that was currently most probable. If they couldn't, then the expert could make a decision without going any further. If they could, then another piece of information would be needed.

Taking the bird/plane example, if we had entered the actual probabilities of the various outcomes instead of letting the expert learn for itself then we would have told the expert that if it had Feathers then it was Bird with probability 1 and it was Plane with probability 0. So, as soon as Feathers turned up (as it were) the expert could calculate that Bird was the only possible outcome and stop asking any more silly questions.

But we haven't proceeded in this way. By example, the expert learns that only outcome Bird has variable Feathers. But it also knows a number of other things. For instance, it knows that outcome Bird has variable Beak as well. Does this mean that it's only a bird if it's got both feathers or a beak? Or is it still a bird if it's only got one of these? And, more to the point, does it mean that in the examples so far it happened by chance that birds had beaks and feathers and that there might yet appear a bird which had neither? In other words, is it a matter of absolute certainty, with probability 1 attached to it, or is it a very high probability, but not 1, which might not be the case later?

Now, as regards birds and planes, you could tell it what to expect. But what about the weather? In this case, nothing occurs with probability 1. It's all pretty uncertain and you aren't in a sufficiently knowledgeable position to explain anything further to the expert system. You don't know the answers yourself, except by waiting to see how the weather eventually does turn out. So how can you explain it to the expert system?

In specific cases a sequential procedure can be devised which will save you time and trouble and which may give as good judgements as a parallel procedure. But, in general, a parallel procedure will make decisions which are as good, and usually better, than a sequential procedure and the program using parallel procedures is much more general purpose because it makes fewer assumptions.

Having said this, let's consider how we might turn our expert system into a sequential device, as shown in Fig. 4.3.

Suppose that it displays the name of the first variable and asks for a value for it which you then provide. It can then calculate the initial D values for the Q outcomes using just this one variable value.

Having done this it can then scan these values of D, select the highest, and consider whether or not that might be the correct outcome. And the way it can do that is by scanning all of the rules in R(I,J) to see if any of them might increase their own D value by enough to exceed the currently-favoured likely outcome. But, to do this involves guessing dummy values of V(I) to work with. Now, if it really guesses it's introducing more uncertainty into the situation and there's enough uncertainty around already. If it's going to do it and be sure of itself it has to know the minimum and maximum values which V(I) can take for each of the variables. If you can provide these values then the expert system can at least discount some options and if it can discount all but one of the options it can make a decision.

But what if it makes an incorrect decision?

Well, if it's been given correct information about the minimum and maximum values of the variables then it can only make an incorrect decision if its rules in R(I,J) are wrong. And it may be that the rules need adjusting on all variables. So the program would have to revert to parallel mode to obtain all of the variable values from you and then adjust R(I,J) as it did in the parallel processing case.

In terms of building your own expert system it may not always be really worth the effort to do all this. Especially if you don't like programming.

But in terms of making the system more interesting or easier to use it might be worth having a go. Whether it will make any difference to what the expert does depends on the minimum and maximum values you

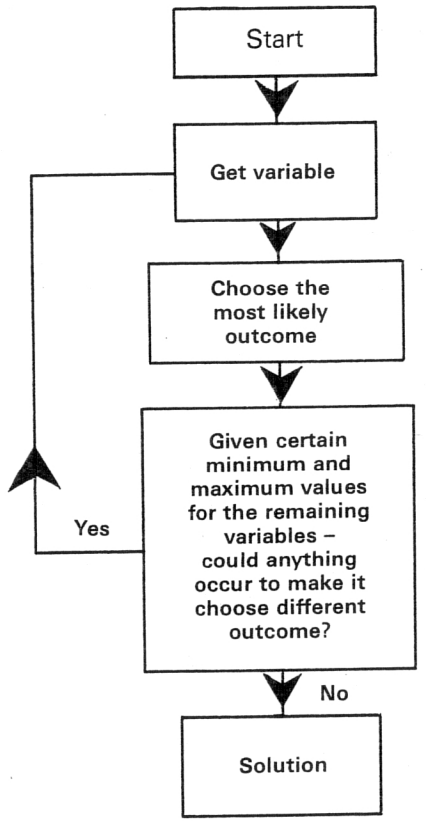


Fig. 4.3
A sequential expert algorithm

provide. They have to be wide enough apart to be genuine limits on the input data - otherwise the expert will make mistakes. But if they're very far apart they won't help the expert to pin the decision down very much and so it will behave just like a parallel processor again - but after a lot more programming effort on your part!

4.2 Adding some commonsense

As we've seen, if you can't tell the expert something more about the problem in hand the parallel procedures we've described give the best solution to date. But if, at the least, you could specify minimum and maximum values for the variables it would be possible for the system, under some circumstances, to skip a few items when asking for information.

Let's outline the method in Applesoft BASIC, step by step. Suppose we DIMension M(2,V) to hold the minimum and maximum values of each of the variables. Minimum values in M(1,I), maximum values in M(2,I). You can arrange to enter these at the start of the program training session either after each variable name is entered or as a separate block of information later on. You also need to DIMension VC(V),PD(V).

Then, as we go into a training session:

```
 2   V = 6:Q = 3
 5   REM :VALUES ARE NEEDED FOR V AND Q
10   DIM VC(V),PD(V),M(2,V),D(Q),V(V),R(V,Q): REM :MINIMUM AND MAXIMUM VALUES
     MUST BE PROVIDED INTO M(2,V)
20   FOR I = 1 TO V
30   VC(I) = 1
40   NEXT
50   REM :THAT SETS UP ARRAY VC AS A FLAG ARRAY TO SHOW WHICH VARIABLES
     HAVE BEEN USED SO FAR IN THE DECISION PROCESS
60   FOR J = 1 TO Q
70   D(J) = 0
80   NEXT
90   REM :CLEAR D
100  FOR VV = 1 TO V
110  INPUT V(VV): REM :GET A VALUE FOR ONE VARIABLE
120  VC(VV) = 0: REM :FLAG SET TO ZERO
130  REM :NOW CALCULATE ARRAY D AS FAR AS POSSIBLE
140  FOR J = 1 TO Q
150  D(J) = D(J) + V(VV) * R(VV,J)
160  NEXT
170  REM  :THAT UPDATES THE DECISION ARRAY
180  D = 0
190  FOR J = 1 TO Q
200  IF D(J) > = D THEN :D = D(J):HJ = J
210  NEXT : REM :FIND THE MAXIMUM D(J) AND PUT IT IN D - HJ IS THE MOST LIKELY
     DECISION TO DATE
220  FOR J = 1 TO Q
230  PD(J) = D(J)
240  NEXT
```



```
250  REM :ARRAY PD HOLDS THE POSSIBLE DECISION VALUES USING THOSE VARIABLES
     THE EXPERT HAS ALREADY GOT
260  FOR J = 1 TO Q
270  FOR I = 1 TO V
280  IF VC(I) = 1 THEN : IF R(I,J) > = R(I,HJ) THEN :PD(J) = PD(J) + (R(I,J) - R(I,HJ)) * M(2,I)
290  IF VC(I) = 1 THEN : IF R(I,J) < R(I,HJ) THEN :PD(J) = PD(J) + (R(I,J) - R(I,HJ)) * M(1,I)
300  NEXT : NEXT
```

(What this does is to go through all of the variables that haven't been used yet. Using the known minimum and maximum values it alters PD(J) in an attempt to find another possible outcome instead of PD(HJ). It does this by assuming that all the remaining variables will act against the current choice. As it only has to find out if the current choice can be displaced it can work with the difference R(I,J)-R(I,HJ) instead of R(I,J) and R(I,HJ) separately.

```
310  H2 = HJ
320  FOR J = 1 TO Q
330  IF PD(J) > = H2 THEN :H2 = PD(J):HI = J
340  NEXT
350  IF HI = HJ THEN : STOP : REM :ALL SOURCES AGREE ON THE OUTCOME
360  IF HI < > HJ THEN : NEXT VV: REM :THERE IS NO AGREEMENT AS YET AND
     ANOTHER VARIABLE VALUE IS NEEDED
```

If, after all this, a wrong decision is still made VC(I) is checked to see which values of V(I) have not been entered yet and these values are provided. The program then goes into its routine to alter R(I,J).

Finding the values of V(I) by working through them in order might not be the method which makes the most sense. For instance, if all values of R(I,J) were equal for a given variable then obtaining a value for that variable would tell the expert system nothing. It might as well skip it. In this case a few lines of code could check, before a variable was requested, if all R(I,J) were equal for that variable. If they are, the expert could jump to the next variable without requesting a value.

What we really need is some general method for deciding which variable value the system should request next. For instance, some variables might be more crucial than others to the outcome. What we have so far is a method which divides the variables into two groups - those which might affect the outcome and those which won't affect the outcome. If we then write routines to try to get values first for those variables which seem likely to affect the outcome most we still have the possibility that, in the end, all of the variables which affect the outcome at all will still be required and so all we've gained is a re-ordering of the variables and not a reduction in the number we've had to consider.

One method which we might try is to look first at those variable values which have the biggest range of variation in terms of their minimum and maximum values (assuming we're able to provide such values). As these variables can vary the most it might be presumed that they contribute the most to the decision-making process. Obviously, in the bird/plane example the minimum and maximum values are 0 and 1 respectively corresponding to the presence or absence of a feature so this wouldn't help us much. However, using data relating to, say, weather conditions we could have variations in minima and maxima for various items. Cloud cover could be nought to ten, corresponding to nought to ten-tenths cloud and rainfall could be nought to two, corresponding to inches of rainfall. So it would seem that cloud cover, with the greatest range, is the item we should check first. Unless, of course, we choose to measure rainfall in millimetres in which case the same rainfall is represented by nought to fifty (millimetres, not inches). And rainfall is the variable to check first.

All this rather serves to illustrate even further the point that, in general, the method of working the expert system can't be pinned down too closely because if it's to be fairly general with regard to the data we give it we can't make it too specific in how it handles that data.

If you build your expert system to be fairly indifferent to the exact area in which it's supposed to acquire expertise then, in specific cases, it's likely to appear a little clumsy and laboured. If that happens then you may feel like customising the thing to a specific application by, say, getting it to ask about particular variables early on in the process and getting it to skip other variables according to some special knowledge of your own. But once you've done that you'll be likely to find that your expert system can't learn other problems quite so readily. It won't be a general-purpose expert system any more even though it might be better for the purpose you built it.

It all really depends on what you want your expert system for. If you know what you want it for then you stand a good chance of being able to build a better one for that purpose than would be possible with any general method given by someone else.

Having said which, we might as well have a crack at getting the system to go for the 'best' variables first. We can do this by considering just how much variation they can cause in the decision rule.

If we DIM RV(V) to hold an indicator of this variation (the 'Rule Values'), we can write:


```
10    FOR I = 1 TO V
20    FOR J = 1 TO Q
30    RV(I) = RV(I) + ABS (M(1,I) - M(2,I)) * ABS (R(I,J))
40    NEXT : NEXT
```

In this, we find the absolute difference between the minimum and maximum values of each variable and multiply this difference by the absolute values of the rules for that variable. Then we would search RV(I) to find that variable which corresponded to the largest value of RV(I). This is the variable that we suppose to be the most important.So we input that value first. If the system calls for another variable it would call for the variable with the next highest value of RV(I). And so on.

The success of this method, apart from the comments made already, depends on the accuracy with which we know the minimum and maximum values for the variables and the extent to which the rules developed by the system so far are good rules.

Obviously, when it's first starting and all R(I,J)=0 it won't have a clue which variables are important to the problem and which aren't. Whether experience will improve the system in this respect depends mainly on the problem it's given to work on.

There is another refinement that could be used when trying to decide which variable to ask about. You'll see that we've got a method of picking that variable with the maximum value in RV(I). Now, suppose that this variable has already been entered. Then VC(I)=0 to indicate that it's no longer needed.

The system then goes off to check if it can deduce an unequivocal outcome. It can't, so it goes looking for another variable, the next largest RV(I).

But, when it was checking for other possible outcomes there might only have been a small number of outcomes which remained possible contenders. Checking the possible decision rule values in PD(J) some of them might have been less than the D(J) the system picked up as the most likely outcome so far. In this case, they aren't possible contenders under any circumstances and they can be ignored for all purposes.

Suppose we DIM QC(Q) for the outcomes and use this array in exactly the same way as we used VC(V) for the variables. Set all of the elements equal to 1 to start off with and set any QC(J) to nought as soon as it becomes apparent that this outcome can never be chosen.

Then, whenever we're dealing with a set of values we can quickly eliminate those which are 'dead'

For instance;

```
10    FOR I = 1 TO V
20    FOR J = 1 TO Q
30    RV(I) = RV(I) + ABS (M(1,I) - M(2,I)) * ABS (R(I,J)) * VC(I) * QC(J)
40    NEXT : NEXT
```

Which has the effect of changing nothing if all of the variables and all of the outcomes are still active because VC(I) and QC(J) are all equal to 1.

But, if any variable has been used, or if any outcome is no longer possible, then VC(I) or QC(J) are equal to 0 and RV(I) is unchanged by that variable or outcome. It simply means that we're only considering the reduced problem of what to do next given that some of the items are no longer active.

Incidentally, there isn't anything sacred about using the values of RV(I) as they're calculated here. You might, for instance want to square R(I,J) instead if using its absolute value. It all depends what you think might be useful. For instance, suppose that you wished to choose between two variables which had two outcomes, and R(IJ) values:

	Outcome 1	Outcome 2
Variable 1	2	2
Variable 2	1	3

Now, if both of these variables had the same maximum and minimum values they would both give the same values in RV(I), because each of them adds up to 4. But you'd like variable 2 to be investigated first because it has the greatest effect (actually, it's the only one that has any effect, in this example!). If you'd taken R(I,J) squared you'd have got;

$RV(1)=2^2+2^2=8$
$RV(2)=1^2+3^2=10$

which would have picked variable 2 first.

Naturally, variable 1 would have been out of the running anyway because it had the same value for each outcome. But suppose you had a third variable:

	Outcome 1	Outcome 2
Variable 3	0	4

This would have made a contribution to RV(3) of the order of 16 and would have been picked before variable 2 (or variable 1). Taking the absolute value of R(I,J) in RV(I) wouldn't have picked up any differences between the three variables although you might have felt that they weren't all making an equal contribution to the problem.

Just as there wasn't anything sacred about the previous method, there's nothing sacred about this one either. Squaring R(I,J) accentuated any differences which might have existed. But taking a higher power would have accentuated them more. Taking an odd power (and not taking the absolute value) would have kept the positive or negative sign of the values intact which might be useful (one can't think why, actually) and taking any other function could make it all behave in a pretty weird fashion which might suit someone's weird program.

Naturally, there's an even more complicated way of doing things but in some cases it might actually be worth doing it.

And that is: to consider the sum of squares about the mean of each variable's rule. To do this you calculate the average value of the rules for each variable and then calculate the squared deviation of the rules about these values. For instance:

```
10    FOR J = 1 TO Q
20    M = M + (R(I,J) / Q) * VC(I) * QC(J)
30    NEXT
40    FOR J = 1 TO Q
50    RV(I) = RV(I) + ((R(I,J) - M) ↑ 2) * ABS (M(1,I) - M(2,I)) * VC(I) * QC(J)
60    NEXT
```

This gives RV(I) as the sum of squares about the mean of R(I,J) for variable I over each of the possible outcomes.

You have to work this out for each I and you should include VC(I) and QC(J) as before when calculating both the means and the sum of squares if you want to allow for variables which have already been given and outcomes which are no longer being considered.

The point about this method is that you might have the following rules:

	Outcome 1	Outcome 2
Variable 1	1	2
Variable 2	1	-2

Intuitively, you feel that variable 2 is the most important because of the large difference it can create in the outcome. But ABS(R(I,J)) wouldn't detect any difference between the two sets and neither would R(I,J) squared. Variable 2's rules vary most about an average value and it's this variation that we can try to measure in order to select the best variable for consideration.

The average value for variable 1 is 1.5 and that for variable 2 is -.5 so we calculate respectively, $(1 - 1.5)^2 + (2 - 1.5)^2 = .5$ and $(1 - (-.5))^2 + (-2 - (-.5))^2 = 4.5$

This method picks variable 2. And, whether it's worth the trouble or not probably depends most on how many variables you have to deal with. If it's a lot you might be glad of any method which helps you get to the right ones first.

4.3 A trial run of our new expert

We will modify our original program so that it will try to eliminate some variables by use of the maximum/minimum method from section 4.2. The modified listing is at the end of this section.

To see how well, or otherwise, the expert works we'll give it the bird/plane/glider example. And, so that we don't get too bogged down, we'll give it three simple pieces of information to work on: wings, beak, engine.

A bird has wings and a beak, but no engine. A plane has wings and an engine but no beak. A glider has wings but no beak or engine.

To start with, the rule array, R(I,J) contains zeros. We tell the expert system that the problem has three variables and three outcomes. We name the variables and the outcomes for it. We then provide minimum and maximum values for the three variables. Clearly, these are nought and one corresponding to the presence or absence of the feature in question.

Variable	Minimum value	Maximum value
Wings	0	1
Beak	0	1
Engine	0	1

At such time as we enter a complete description for any one of the three outcomes the description would be held in V(I) and would be:

Variable	Bird	Plane	Glider
Wings	1	1	1
Beak	1	0	0
Engine	0	1	0

The order of events is now up to the user (you) but we'll suppose that it happened like this.....

We think of a bird. And we input V(I) values to correspond to a bird. The rule array R(I,J) is adjusted because the expert makes an incorrect guess:

R(I,J)	Bird	Plane	Glider
Wings	1	-1	-1
Beak	1	-1	-1
Engine	0	0	0

In other words, V(I) for a bird has been added to the outcome bird rule and subtracted from the other rules.

The expert now enters another session. We think of a bird again.

It first asks for a value for 'Wings'. We reply with 1.

The system then asks if it can deduce outcome bird. Which it can. We reply, Yes, and nothing is adjusted.

Now, obviously, we know that wings aren't sufficient to indicate a bird. It's really no evidence at all. But in the current state of knowledge of the expert it is enough. For with V(I)=(1,x,x) the rules for a bird give a bigger value than the rules for any other outcome. And, considering the minimum and maximum information the system has, no values exist for the last two variables in V(I) which could ever change this decision.

So, the expert has got the right answer on imperfect information, not by working incorrectly but because its set of rules are, so far, imperfect. We might show a little charity towards it at this stage - after all, it's only ever come across one bird in its whole life so far.

Now another session. We think of a plane.

It asks for a value for 'Wings'. We reply with 1.

Again the system asks if it can deduce outcome bird. Which is a reasonable question to ask because, last time, it could. We reply that it can't.

The system asks which outcome it should have been. We reply Plane.

The system now asks for values for the rest of V(I) so that it can readjust its rules. We give it the correct values for beak and engine so V(I)=(1,0,1).

The rules are adjusted.

Now another session. We think of a plane again.

It first asks for a value for 'Beak'. We reply with 0. It then asks about 'Engine'. We reply with 1.

The system now asks if it can deduce outcome plane. Which it can.

Now this is quite promising. It didn't ask about wings this time which is a good thing because it wouldn't have learned anything by the question.

Let's look at the rule array, R(I,J) as it now exists:

	Bird	Plane	Glider
Wings	0	0	-2
Beak	1	-1	-1
Engine	-1	1	-1

After the last mistake it made, the expert added the plane variables to the plane rule and subtracted them from the bird and glider rules.

Now: why did it ask about beak first?

It did it because this seemed like the most important variable of all. In each rule there is a 1 or -1 against beak so, summing the products of the maximum variable values and the absolute values in the rule array, the expert rates beak as 3, engine as 3 and wings as 2 in order of importance. Beak comes before engine in the list of variables so it asks about beak first.

From now on, if we just consider birds and planes, the expert will always be right. Every time it will ask about the object's beak. If it's got one it will judge it to be a bird. If it hasn't, it will ask about the engine and, if it has one of those, it will judge it to be a plane.

Now another session. This time we think of a glider.

In response to a query about the object's beak we reply with '0'.It then asks about 'engine' - reply '0'. And 'wings' - reply '1'.

At this point it guesses that it's another plane - which it isn't and we tell it so. We tell it that this is a glider and it adjusts array R again.

Another session. This time we think of a bird.

The expert asks about wings and beaks. We reply '1' and '1' and it then correctly, decides: bird.

Another session. We think of a glider. It asks about wings, beak and engine we reply with '1', '0', '0'. The expert guesses glider - which is right.

Another session and this time we think of a plane. It asks about wings, beak and engine and we reply '1', '0', '1' and it deduces, correctly, that this is a plane.

Another session, and this time we give it the bird. It asks about wings and beak only and we reply '1', '1' - and it gets it right.

After this, it's perfect and makes no more mistakes. It can correctly identify a bird, plane or glider and it doesn't always need to ask about all three variables to do it.

The set of rules at this stage is:

	Bird	Plane	Glider
Wings	-1	-1	-1
Beak	1	-1	-1
Engine	-1	1	-1

There are a few points to make about these rules at this stage.

First, it only guesses glider correctly by, as it were, default.

That is: glider has the variable values (1,0,0,) - so any of the three outcomes gives a decision value of -1, there is no reason for picking glider against any other outcome. The reason it does pick glider correctly is that, having found the object has wings and no beak, it searches for the maximum decision value. They are all equal so it comes to rest on glider - the last value it looks at. When it finds the object has no engine either it

does the same thing - coming to rest on glider as the last item in the list and finding that nothing can displace this judgement by being greater in value than this.

In an example like this a lot depends on whether rules are adjusted and decisions made on the basis of a conditional test of 'greater than' or greater than or equal to'. The more stringent test (greater than or equal to) alters the rules more often and, initially, makes the most mistakes. But if often results in a slightly better rule set - one which asks fewer questions when it is possible to do so. A simple 'greater than' test alters the rules less often and works, in part, because of the natural order of the data in the list - which is something you should remember when you are trying to work out what your program has done with its rules.

A further point is that the expert often asks about Wings - and we know that this question has no bearing on the outcome at all. The reason it does this is because the rule values in array RV are (3,3,3) for (wings, beak, engine) - and it's this that makes us think that some other method of calculating the rule values might have been preferable. For instance, if we calculated the means for each variable (-1, -⅓, -⅓) we could have calculated the sum of squares about the mean of each variable (0, 2.67, 2.67) - which gives beak and engine as, clearly, the most important variables and leaves wings right out, as it were, in the cold. (The sum of squares of a variable about its mean is, incidentally, the same as the variance of that variable but without dividing by n - see the Technical Overview at the back if you're not sure about this.)

Also, we'd have been quite likely to get a different set of rules if we'd trained the expert by giving it the problems in a different order.

But all that we require of a decent set of rules is that they should work - which these do.

Think about our talk about linear separability. We've got an expert system which has managed to draw lines between the three groups of objects - birds, planes and gliders. It doesn't matter exactly where the lines are drawn as long as they serve to separate the three groups. The current set of rules does that but it wouldn't be hard to think of another set of rules which did it just as well.

This example is fairly neat - but don't suppose that every problem given to the expert will look as nice as this one does.

Suppose that we keep our three outcomes: bird, plane, glider. Now give it

six variables to work on: wings, tail, beak, feathers, engine, undercarriage. Commonsense tells us that most of these variables are redundant - they aren't needed to answer the basic question. But will the expert work that out?

The answer is: No.

As there are still only three outcomes we know that no more than two well-chosen questions will still settle the matter. However, after the expert had been sufficiently trained, it asked about the following items for each of the three outcomes:

Bird	Engine - Reply 0 Undercarriage - Reply 0 Beak - Reply 1	Deduction: Bird
Plane	Engine- Reply 1 Undercarriage - Reply 1	Deduction: Plane
Glider	Engine - Reply 0 Undercarriage - Reply 0 Beak - Reply 0 Feathers - Reply 0	Deduction: Glider

Now this isn't bad. It gets it right every time and it doesn't ask about all of the variables. For instance, it never asks about wings and tail - which is good, because these variables don't tell it anything. However, from our point of view, it's rather dense in asking about the undercarriage straight after asking about the engine every time. But that's because we know that, in these examples, the two always go together. Likewise, it often asks about feathers straight after asking about the beak - another two items which we know always go together.

The point is, of course, that the expert system doesn't know that these things always go together. Having asked about the engine, a question it thinks is important, it finds that it can't be sure of the outcome and so it has to ask another question. The undercarriage seems important to it so it asks about that. It didn't realise that it could have deduced undercarriage from engine. For all the expert knew, engine and undercarriage might have just occurred together by chance so far. In the problem it was working on at that time engine and undercarriage might not have occurred together and that would have changed its view of the possible outcome. It simply needed to keep on asking questions until, given the state of its rules and

the minimum and maximum values assigned to each variable, there was absolutely no possibility whatsoever of its getting fresh information on any variable which would cause it to change its mind about the likely outcome.

Knowing that the object had an engine wasn't enough for it to be absolutely sure of the outcome. Suppose (you can imagine it thinking to itself) I had an object with an engine and then it suddenly turned out to have no undercarriage. Would that cause me to think differently about what it was? Yes, it would. Better ask about the undercarriage.

Precisely what sort of object the expert thought might have an engine and no undercarriage is a bit of a mystery. Maybe we should have included the outcome: rocket. Just to keep the expert happy.

The following is a listing of the program used in this section. It uses the learning algorithm. Minimum and maximum values are used in order to try to come to a speedy conclusion. Questions are asked on the basis of their values - using the ABS function with the min/max values to try to assess their importance.
Outcomes which cannot achieve a high enough value to displace the current best guess are eliminated from further calculations.

Once you've run this try altering the code so that the rule values in RV are calculated using the sum of squares about their means (see p. 77) rather than the ABS function - it should influence the order of question-asking.

Fig. 4.4
Minimum/maximum modified program

Apple II listing

```
 20  HOME : INPUT "HOW MANY VARIABLES HAVE YOU ?";V
 30  DIM V(V),V$(V),M(2,V),VC(V),PD(V),RV(V)
 40  PRINT "PLEASE NAME THESE VARIABLES"
 50  FOR I = 1 TO V
 60  PRINT "VARIABLE ";I;" IS ";: INPUT V$(I)
 70  INPUT "IT HAS MINIMUM VALUE = ";M(1,I): INPUT "AND   MAXIMUM VALUE = ";M(2,I)
 80  NEXT
 90  INPUT "HOW MANY OUTCOMES HAVE YOU ?";Q: DIM Q$(Q),R(V,Q),D(Q),QC(Q)
100  PRINT "PLEASE NAME THESE OUTCOMES"
110  FOR I = 1 TO Q
120  PRINT "OUTCOME ";I;" IS ";: INPUT Q$(I)
130  NEXT
140  PRINT "VARIABLE","MIN VALUE"; SPC( 5);"MAX VALUE": FOR I = 1 TO V: PRINT
     V$(I),M(1,I),M(2,I): NEXT : PRINT "PRESS ANY KEY TO CONTINUE": GET X$
```



```
150  HOME : PRINT "THIS IS A TRAINING SESSION": PRINT "PROVIDE VALUES OF
     VARIABLES": PRINT "I WILL GUESS AN OUTCOME": PRINT "YOU MUST TELL ME IF I AM
     RIGHT OR WRONG"
160  FOR I = 1 TO V:VC(I) = 1:V(I) = 0: NEXT : FOR J = 1 TO Q:D(J) = 0:PD(J) = 0:QC(J) = 1:
     NEXT
170  RV = 0:VV = 1: FOR I = 1 TO V:RV(I) = 0: FOR J = 1 TO Q:RV(I) = RV(I) + ABS (M(1,I) -
     M(2,I)) * ABS (R(I,J)) * VC(I) * QC(J): NEXT : IF RV(I) > RV THEN :VV = I:RV = RV(I)
180  NEXT : REM :THAT PICKED A PROMISING VARIABLE TO LOOK AT
190  IF RV = 0 THEN : GOTO 390: REM :THE MOST PROMISING VARIABLE WOULD
     CONTRIBUTE NOTHING
200  PRINT "VARIABLE ";VV;" (";V$(VV);") IS ";: INPUT V(VV)
210  VC(VV) = 0: REM  :SET FLAG TO ZERO
220  FOR J = 1 TO Q
230  D(J) = D(J) + V(VV) * R(VV,J)
240  NEXT : REM :THAT UPDATES THE DECISION ARRAY
250  D = D(1): FOR J = 1 TO Q
260  IF D(J) > = D THEN :D = D(J):HJ = J
270  PD(J) = D(J): NEXT : REM :THAT FINDS THE BEST GUESS TO DATE,HJ
280  FOR J = 1 TO Q
290  FOR I = 1 TO V
300  IF VC(I) = 1 THEN : IF R(I,J) > R(I,HJ) THEN :PD(J) = PD(J) + (R(I,J) - R(I,HJ)) * M(2,I)
310  IF VC(I) = 1 THEN : IF R(I,J) < R(I,HJ) THEN :PD(J) = PD(J) + (R(I,J) - R(I,HJ)) * M(1,I)
320  NEXT : NEXT : REM :THAT GETS ANY POSSIBLE ALTERNATIVE VALUES
330  H2 = PD(HJ):HI = HJ: FOR J = 1 TO Q
340  IF PD(J) > H2 THEN :H2 = PD(J):HI = J
350  IF PD(J) < D(HJ) THEN :QC(J) = 0: PRINT "IT CAN'T BE ";Q$(J): REM  :THIS OUTCOME IS
     NO LONGER LIKELY
360  NEXT
370  IF PD(HI) > PD(HJ) THEN : GOTO 170: REM - THERE'S STILL UNCERTAINTY SO GET
     ANOTHER VARIABLE
380  PRINT "IS THE OUTCOME ";Q$(HI);" ?": INPUT A$: IF A$ = "Y" THEN : PRINT "WHOOPEE
     !": PRINT "PRESS ANY KEY TO CONTINUE": GET X$: GOTO 150
390  FOR I = 1 TO Q: PRINT I;" ";Q$(I): NEXT : INPUT "WHICH OUTCOME IS IT ?";P
400  FOR J = 1 TO V: IF VC(J) = 1 THEN : PRINT "WHAT WAS THE VALUE OF VARIABLE ";J;"
     (";V$(J);") ?";: INPUT V(J):VC(J) = 0
410  NEXT
420  FOR I = 1 TO Q: IF D(I) > = D(P) AND I < > P THEN : FOR J = 1 TO V:R(J,I) = R(J,I) - V(J):
     NEXT
430  NEXT
440  FOR J = 1 TO V:R(J,P) = R(J,P) + V(J): NEXT
450  PRINT "THE CURRENT RULE ARRAY IS :-"
460  PRINT  SPC( 10);: FOR J = 1 TO Q: PRINT Q$(J); SPC( 10 -  LEN (Q$(J)));: NEXT
470  FOR I = 1 TO V: PRINT V$(I); TAB( 10);: FOR J = 1 TO Q: PRINT R(I,J); TAB( 10 + J * 20);:
     NEXT : PRINT : NEXT
480  PRINT "PRESS ANY KEY TO CONTINUE": GET X$: GOTO 150
```

```
  2 REM Note that the arrays ar
e renamed for the Spectrum as it
 only permits single letters
  10 CLS
  20 INPUT "How many variables?"
;v
  25 DIM v(v): DIM v$(v,10): DIM
 m(2,v): DIM c(v): DIM p(v): DIM
 a(v)
  30 PRINT AT 16,0;"Please name
them:"
  40 FOR i=1 TO v: INPUT "Variab
le ";(i);" is ";v$(i)
  45 INPUT "It has minimum value
=";m(1,i): INPUT "And maximum va
lue=";m(2,i)
  50 NEXT i
  55 CLS
  70 INPUT "How many outcomes ha
ve you?";q
  75 DIM q$(q,20): DIM r(v,q): D
IM d(q): DIM q(q)
  80 PRINT AT 16,0;"Please name
them"
  90 FOR i=1 TO q: INPUT "Outcom
e ";(i);" is ";q$(i): NEXT i
 100 CLS : PRINT "Variable";TAB
 12;"Minimum";TAB 22;"Maximum": P
RINT : FOR i=1 TO v: PRINT v$(i)
;TAB 12;m(1,i);TAB 22;m(2,i): NE
XT i: PRINT : PRINT "Press any k
ey to continue"
 105 LET a$=INKEY$: IF a$="" THE
N GO TO 105
 110 CLS
 120 PRINT "This is a training s
ession": PRINT : PRINT "Please p
rovide values of ": PRINT "varia
bles": PRINT : PRINT "I will gue
ss an outcome": PRINT : PRINT "T
ell me if I am correct"
 125 PRINT : PRINT "Press a key
to continue"
 126 LET a$=INKEY$: IF a$="" THE
N GO TO 126
 128 CLS : PRINT "Tell me your v
ariables"
 130 FOR i=1 TO v: LET c(i)=1: L
ET v(i)=0: NEXT i: FOR j=1 TO q:
 LET d(j)=0: LET p(j)=0: LET q(j
)=1: NEXT j
```



```
 160 LET rv=0: LET vv=1: FOR i=1
 TO v: LET a(i)=0: FOR j=1 TO q:
 LET a(i)=a(i)+ABS (m(1,i)-m(2,i
))*ABS (r(i,j))*c(i)*q(j): NEXT
j: IF a(i)>rv THEN LET vv=i: LET
 rv=a(i)
 180 NEXT i: REM That picked a P
romising value
 185 IF rv>0 THEN GO TO 200
 190 PRINT : PRINT "At this stag
e, you will";"have to tell me th
e outcome",,
 192 PRINT "Press a key to conti
nue"
 194 LET a$=INKEY$: IF a$="" THE
N GO TO 194
 196 GO TO 390
 200 INPUT "Number ";(vv);" (";(
v$(vv));") is ";v(vv)
 210 LET c(vv)=0: REM set flag t
o zero
 220 FOR j=1 TO q
 230 LET d(j)=d(j)+v(vv)*r(vv,j)
 240 NEXT j: REM Update decision
 array
 250 LET d=d(1): FOR j=1 TO q
 260 IF d(j)>=d THEN LET d=d(j):
 LET hj=j
 270 LET p(j)=d(j): NEXT j: REM
that finds the best guess to dat
e,hj
 280 FOR j=1 TO q
 290 FOR i=1 TO v
 300 IF c(i)=1 THEN : IF r(i,j)>
r(i,hj) THEN LET p(j)=p(j)+(r(i,
j)-r(i,hj))*m(2,i)
 310 IF c(i)=1 THEN : IF r(i,j)<
r(i,hj) THEN LET p(j)=p(j)+(r(i,
j)-r(i,hj))*m(1,i)
 320 NEXT i: NEXT j: REM  that g
ets any possible alternatives
 330 LET h2=p(hj): LET hi=hj: FO
R j=1 TO q
 340 IF p(j)>h2 THEN LET h2=p(j)
: LET hi=j
 350 IF p(j)<d(hj) THEN LET q(j)
=0: PRINT "It can't be ";(q$(j))
: REM this is no longer likely
 360 NEXT j
 370 IF p(hi)>p(hj) THEN GO TO 1
60: REM get another variable
 380 INPUT "Is the outcome ";(q$
(hi));"?";a$: IF a$="y" THEN PRI
NT ,,,"WHOOPEE!"
 381 IF a$<>"y" THEN GO TO 390
 382 FOR i=1 TO 100: NEXT i: PRI
NT "Press any key to continue"
 385 LET a$=INKEY$: IF a$="" THE
N GO TO 385
 386 GO TO 128
```

```
 390 CLS : FOR i=1 TO q: PRINT i
;" ";(q$(i)): NEXT i: INPUT "Whi
ch outcome is it?";p
 400 CLS : FOR j=1 TO v: IF c(j)
=1 THEN PRINT AT 16,0;"What was
the value"'"of variable ";(j);"
(";(v$(j));")  ?": INPUT v(j): LE
T c(j)=0
 410 NEXT j
 420 FOR i=1 TO q: IF d(i)>=d(p)
 AND i<>p THEN : FOR j=1 TO v: L
ET r(j,i)=r(j,i)-v(j): NEXT j
 430 NEXT i
 440 FOR j=1 TO v: LET r(j,p)=r(
j,p)+v(j): NEXT j
 450 CLS : PRINT "The current ru
le array is as follows for each
outcome"
 460 PRINT : PRINT
 470 FOR j=1 TO q
 480 PRINT q$(j): PRINT
 490 FOR i=1 TO v
 500 PRINT v$(i),r(i,j)
 510 NEXT i
 520 PRINT "Press any key to con
tinue"
 530 LET a$=INKEY$: IF a$="" THE
N GO TO 530
 540 CLS : NEXT j
 550 GO TO 128
```


Chapter 5

A Real World Expert

5.1 The weather again

It's all very well to have an expert system which can answer contrived questions about birds, planes and gliders. It makes a nice toy. But what happens if we give it a real problem such as the weather? What will it do if we ask it if it's going to rain tomorrow?

Well, it's a real problem certainly. We don't know how to answer that question ourselves so if the expert can work something out for us we might feel that we'd got something here.

So, forget contrived examples. We'll turn to the London Weather Centre for some real data.

Each month the London Weather Centre publishes figures showing the rainfall, temperature and sunshine for each day. By looking at any given day we could provide data for the expert and train it by letting it know if it was raining on the following day or not.

As we progressed through the records we (might) find that our expert's guesses improve.

The figures shown in Table 5.1 are for March 1982 and give minimum and maximum temperatures (in degrees centigrade), rainfall (in millimetres), and sunshine (in hours). If the rainfall is zero it's (would you believe) a dry day. Otherwise, we'll consider it to be raining.

We can provide some minimum and maximum values, although it wouldn't be too hard to find weather that fell outside these ranges:

Temperature:	minimum 0°C.	maximum 20°C
Rainfall:	minimum 0 mm	maximum 25 mm
Sunshine:	minimum 0 hours	maximum 12 hours

Day of Month	Min. Temp. °C.	Max. Temp. °C.	Rainfall mm.	Sunshine hours
1	9.4	11.0	17.5	3.2
2	4.2	12.5	4.1	6.2
3	7.6	11.2	7.7	1.1
4	5.7	10.5	1.8	4.3
5	3.0	12.0		9.5
6	4.4	9.6		3.5
7	4.8	9.4		10.1
8	1.8	9.2	5.5	7.8
9	2.4	10.2	4.8	4.1
10	5.5	12.7	4.2	3.8
11	3.7	10.9	4.4	9.2
12	5.9	10.0	4.8	7.1
13	3.0	11.9	0.2	8.3
14	5.4	12.1		1.8
15	8.8	9.1	8.8	
16	2.4	8.5	3.0	3.1
17	4.3	10.8	4.2	4.3
18	3.4	11.1		6.6
19	4.4	8.4	5.4	0.7
20	5.1	7.9	3.0	0.1
21	4.4	7.3	1.0	
22	5.6	14.0		6.8
23	5.7	14.0		8.8
24	2.9	13.9		9.5
25	5.8	16.4		10.3
26	3.9	17.0		9.9
27	3.8	18.3		8.3
28	5.8	15.4		7.0
29	6.7	8.8	6.4	4.2
30	4.5	9.6		8.8
31	4.6	9.6	3.2	4.2

Table 5.1
London weather summary for March 1982.
(A gap indicates zero)

Apart from the minimum rainfall and sunshine figures these minimum and maximum values would have to be widened to allow for weather all the year round.

Incidentally, it was the sunniest March since 1967 and the second sunniest March since records began in 1929 according to the Meteorological Office who are well-known optimists and who even extend their optimism to telling people that what we've just had was good weather.

Anyway, we set up the expert system with four variables: Minimum temperature, maximum temperature, rainfall and sunshine.

We give it what we reckon are reasonable minimum and maximum values.

We give it two outcomes: Rain and No Rain.

The first item is: 1st March 1982. We enter the variables and let the expert know that there was rain the following day.

Then it is the next day. We give it the values for the 2nd March and, if it guesses wrong about the next day's weather we let it know that, again, it rained the following day.

Then, it is the next day and, yes, you are Right! Excitement is not a feature of this process.

But if you glance at Table 5.2 you'll see that there are 18 wet days and 13 dry days in March 1982. Which suggests that there's a fifty-fifty (roughly) chance of getting the answer right just due to chance. How, then, does the expert get on - and is this any better than chance?

From Table 5.2 , you can see that the expert predicted rain, or its absence, correctly for 22 of the 30 days (the first day we don't count because the expert had no rules established until it knew the outcome of day one). This is 73 per cent success and actually sounds pretty good. Possibly much better than a human would have guessed. But let's go over the results in a bit more detail.

Day of Month	Actual Weather next day	Weather Forecast by expert
1	Rain	Don't know
2	Rain	Rain
3	Rain	Rain
4	Dry	Rain - Error
5	Dry	Rain - Error
6	Dry	Dry
7	Rain	Dry - Error
8	Rain	Rain
9	Rain	Rain
10	Rain	Rain
11	Rain	Rain
12	Rain	Rain
13	Dry	Dry
14	Rain	Rain
15	Rain	Rain
16	Rain	Rain
17	Dry	Rain - Error
18	Rain	Dry - Error
19	Rain	Rain
20	Rain	Rain
21	Dry	Rain - Error
22	Dry	Dry
23	Dry	Dry
24	Dry	Dry
25	Dry	Dry
26	Dry	Dry
27	Dry	Dry
28	Rain	Dry - Error
29	Dry	Rain - Error
30	Rain	Rain
31	Rain	Rain

Table 5.2
Actual & predicted weather for March 1982, London

The first thing to do is to damp down a bit of the (possible) enthusiasm for the results. We do this by pointing out that there is a very simple rule which, with this data, would have made ten mistakes over the whole month. This rule is: that the weather tomorrow will be the same (rain or no rain) as it is today. If we use this rule we'll only be wrong every time a dry day precedes a wet day or a wet day precedes a dry day. By and large, using this rule, we would expect to be right most of the time simply because we know that weather comes in runs - we tend to have a spell of dry days followed by a spell of wet days.

So, by this line of reasoning, if all the expert system did was to adjust its rule base to predict for tomorrow what we had for today then it would have made only ten mistakes. In fact, it's done better than that with only eight mistakes - which shows that it hasn't been quite so simple. But, if you look at the results you'll see that it was when the weather actually did change that the expert was caught out. It wasn't always caught out by a change in the weather, as you can see on the 13th March, but it generally was.

What happens is that, after the first day, the expert makes predictions of, in this case, rain. These are correct until (in case you hadn't guessed) they are wrong - due to a dry day occuring. This error causes expert to adjust its rules and it eventually gets itself sufficiently sorted out to predict a dry day on the 6th March.

At this point the weather turns wet again and the expert makes a mistake. By this time it has adjusted its rules three times and these rules give it pretty good service until the 17th of the month when a dry day arrives unexpectedly. Some more adjustment occurs to the rules and, by the 22nd, the expert is running well enough to predict a dry spell. On the 28th and 29th more errors occur as rain returns.

In one way of thinking our expert has performed poorly because it does periodically make mistakes. It isn't perfect. And what will concern some people is the fact that, as it adjusts its rule base every time it makes a mistake, it isn't using the same set of rules throughout the performance which might cause one to think that, in some sense, it's cheating. After all, you may say, what is it that we're judging here? With most games you aren't allowed to change the rules every time you lose.

It's this changing of the rules that makes our expert difficult to analyse except by trying it out to see what it does. If it had one set of rules which never changed then it would be possible to say more about them - but it doesn't, it just has a general method for supplying an answer.

In some ways, this makes the system more human. After all, if you were asked what you thought the weather would do tomorrow you'd probably take account of what the weather had been doing recently. By this, one doesn't mean solely what the weather has done today. If, by and large, the weather continues dry a light shower isn't going to cause you to change your mind about the chance of a dry day tomorrow. Or, at any rate, it's unlikely to. You'll approach the matter of weather prediction with a given mental 'set', or predisposition, to interpret today's clues in the context of generally recent weather.

Suppose, for instance, that you wanted to guess if it would snow tomorrow. Now, if it were the middle of January and there'd been a lot of snow recently and there were signs today of more snow to come then you'd be fairly happy to predict snow. But, if it were the middle of July and there were signs which, in mid-January, would indicate snow then you'd be more reluctant to predict snow. The fact is: that you're using different rules yourself at different times.

And if, in mid-July, it actually did snow then you'd quickly adjust your internal weather prediction rules and be much more willing to reckon on the possibility of snow tomorrow if the same signs appeared again. And this is what our expert does. Having said which, its behaviour begins to look a lot more reasonable.

Also looking reasonable is its behaviour in asking for information. Recall that our expert is free to choose which variables it first enquires about and free to make a prediction without asking about all variables if it's sure of its outcome.

In this example it makes use of this freedom. Almost invariably the one question it asks first is: rainfall?

Having got a figure for today's rainfall it then, usually, makes a decision without checking any other items and, as we've seen, it's usually right.

What it appears to be doing is predicting rain when today's rainfall is high and predicting dry weather when today's rainfall is low. Just how high is high and how low is low varies though. And, in between, there's a grey area in which it just can't make up its mind. In this grey area it will sometimes call on the other variables to add to the rainfall information, typically asking about temperature and sunshine. Very rarely does it want to know values for all the variables involved.

In all, its behaviour seems reasonable. Also reasonable is the set of rules which it has developed by the end of the month:
The array R as of 31st March 1982:

	Rain Tomorrow	Dry Tomorrow
Minimum Temperature	-0.6999	0.6999
Maximum Temperature	-2.5	2.5
Rainfall	4.1	-4.1
Sunshine	4.6	-4.6

We can see that, for a prediction of rain tomorrow, the expert will look for high rainfall figures today with low temperatures. Strangely, it will also look for high sunshine figures as a predictor of rain - but, maybe, this isn't so strange. Cold, wet and sunny weather is certainly 'unsettled' weather - so, maybe, that's what the expert is looking for. After all, by the end of March we might well expect some sunny April showers to be starting!

On the other hand, for a dry day tomorrow the expert is looking for warm, dry, overcast weather today - maybe this is the sort of still weather that doesn't presage rain. In all, these are not items which we would automatically use ourselves to predict the weather tomorrow - but maybe we should do. If our expert system can make a good prediction of the weather in this way it may well be that, by now, it genuinely does have some expertise which exceeds our own in some respects. Possibly, having initially learnt from us we can now learn from it.

But just how well has it done?

Actually, we can be a bit more precise about how well it's done by saying that, not only does it get things right 73 per cent of the time (for this example) but that the probability of this being due to chance is less than 0.025 - i.e. less than 25 chances in one thousand. And how, you may wonder, do we say such a thing as this?

Well, a bit of statistics comes in. We say that there have been 17 rainy days and 13 dry days in March and the expert has predicted 19 rainy days and 11 dry days. We can represent this as follows:

Expert prediction

		Rain	Dry		Total
Actual	Rain	14	3	=	17
Weather	Dry	5	8	=	13
	Total	19	11	=	30

On 14 days rain was predicted and there was rain. On 8 days a dry day was predicted and it was dry. On 8 days mistakes were made.

Using the totals we can calculate what those figures would have been by chance. These are the so-called 'expected values' and a description of how to calculate them is given at the end of this section:

Expected values

		Rain	Dry		Total
Actual	Rain	10.77	6.23	=	17
Weather	Dry	8.23	4.77	=	13
	Total	19	11	=	30

We can use these two tables in a statistical test called a chi-squared test.

A chi-squared test of significance says that the table built up from the actual predictions of the expert is different from the table based on chance and that it is so different that the probability that expert is working by chance is less than 0.025.

The problem with that sort of statement is clearly that: you may not know what a chi-square test is or whether to believe it or not.

Well, you can take it on trust if you like and believe it.

If you actually want to know how it works...

The first table contains the Observed frequencies, we'll call these O (for 'observed'). The second table contains the expected frequencies and we'll call them E (for 'expected').

Chi-square is calculated by:

CH = CH + ((O-E) ↑2)/E

for each of the four cells in the table.
So:

CH = ((14-10.77)↑2)/10.77) + ((3-6.23)↑2)/6.23) + ((5-8.23)↑2)/8.23)
+ ((8-4.77)↑2)/4.77, for this example.

So: CH = 6.11

Now, you look up the value of chi-square in a table of statistics (sorry, you'll have to buy a table of statistics). Look against the values for 1 degree of freedom and you'll see that under the 0.25 (some call it 2.5 per cent) column, chi-square has a value of 5.02. Our value of chi-square is bigger than this so we can place at least .025 probability confidence in the possibility that our results are not due to chance.

Some of you will find this all a bit tedious so you can always ignore it. But it does give you a way of judging the performance of your expert system in a fairly exact fashion. After all, you may have a different problem and may want to estimate how well your expert is at working on that.

You may also, if you're interested, wonder what degrees of freedom are and how we came by the expected values.

Degrees of freedom depend on the number of rows and columns in the table. We calculate:

DF= (R-1)*(C-1), where R= rows and C= columns, and DF= degrees of freedom.

In this case R=2, C=2, so DF=1. But you might have had more outcomes and, so, a bigger table.

Expected values are calculated for each cell by:

Looking to see what the row total is for that cell. (Observed).

Looking to see what the column total is for that cell. (Observed).

Multiplying these two together.

Dividing them by the overall total. (Observed).

Repeating this process for each cell.

Entering these values into the Expected Values table.

If you've done it right the Expected table should have the same row, column, and overall totals as the Observed table.

In our example the first cell in the expected table was 10.77 because: 10.77 = 17 X 19/30.

Actually, it's an ideal subject for a computer program, if you're still interested and awake by now.

If you don't happen to have any statistical tables handy then you could just note that if chi-square equals nought then the results could certainly be due to chance. The bigger chi-squared, the better for our purposes.

5.2 A Chi-squared program

If you feel interested in testing experts' results in a formal way using chi-squared, but haven't come across the method before, then you might as well have a program to do it for you.

You could jot down your results, as before, into a table, and then follow the logic of the following Applesoft program.

```
10    HOME : INPUT "HOW MANY ROWS HAVE YOU ?";R
20    INPUT "HOW MANY COLUMNS HAVE YOU?";C
30    DIM Q(R + 1,C + 1), E(R,C)
40    FOR I = 1 TO R
50    FOR J = 1 TO C
60    PRINT "ROW ";I;" COLUMN ";J;" = ";: INPUT Q(I,J)
70    NEXT : NEXT
80    FOR I = 1 TO R
90    FOR J = 1 TO C
100   Q(R + 1,J) = Q(R + 1,J) + Q(I,J)
110   Q(I,C + 1) = Q(I,C + 1) + Q(I,J)
120   Q(R + 1,C + 1) = Q(R + 1,C + 1) + Q(I,J)
130   NEXT : NEXT
140   FOR I = 1 TO R
150   FOR J = 1 TO C
160   E(I,J) = Q(I,C + 1) * Q(R + 1,J) / Q(R + 1,C + 1)
170   NEXT : NEXT
180   FOR I = 1 TO R
190   FOR J = 1 TO C
200   CH = CH + ((Q(I,J) - E(I,J)) ↑ 2) / E(I,J)
210   NEXT : NEXT
220   DF = (R - 1) * (C - 1)
230   PRINT "CHI SQUARE = ";CH;" DEGREES OF FREEDOM = ";DF
```

This prints the answers. And, sorry, but you still need those tables!

The Sinclair Spectrum equivalent program is shown below:

```
  10 CLS : INPUT "How many rows?
";r
  20 INPUT "How many columns?";c
  30 DIM q(r+1,c+1): DIM e(r,c)
  40 FOR i=1 TO r
  50 FOR j=1 TO c
  60 PRINT AT 16,0;"Row ";i;" Co
lumn ";j;" is?"
  65 INPUT q(i,j)
  70 NEXT j: NEXT i
  80 FOR i=1 TO r
  90 FOR j=1 TO c
 100 LET q(r+1,j)=q(r+1,j)+q(i,j
)
 110 LET q(i,c+1)=q(i,c+1)+q(i,j
)
 120 LET q(r+1,c+1)=q(r+1,c+1)+q
(i,j)
 130 NEXT j: NEXT i
 140 FOR i=1 TO r
 150 FOR j=1 TO c
 160 LET e(i,j)=q(i,c+1)*q(r+1,j
)/q(r+1,c+1)
 170 NEXT j: NEXT i
 175 LET ch=0
 180 FOR i=1 TO r
 190 FOR j=1 TO c
 200 LET ch=ch+(ABS (q(i,j)-e(i,
j))↑2)/e(i,j)
 210 NEXT j: NEXT i
 220 LET df=(r-1)*(c-1)
 230 CLS : PRINT AT 20,0;"Chi Sq
uare= ";ch'"Degrees of Freedom=
";df
```

5.3 Exercising your expert

There is, at this stage, one little point which, in fairness, ought to be made about your expert system.

It's, in many ways, fine. There's nothing wrong with it. It will work. It will learn its expertise with little or no help from you. It only needs to be given a few examples to start with.

But how many examples does it need? At what point will it be making expert judgements of the very best quality?

Well... Considering the bird/plane/glider example you might innocently suppose that three examples would do. One of each sort. After all, it will have seen everything once it's been shown all three. True, it will have seen everything. But not true that it will have developed adequate rules to recognise them again. It will take it rather more than three attempts to shuffle its set of rules around until it's got a really decent method of working.

So, to be precise, just how many examples does it need to be given. In actual numbers.

Well (and at this point one sort of shuffles the feet a bit and attempts, as it were, to avoid the reader's eye) the theory actually depends on the expert having access to one example of each possible input actually, well, an infinite number of times, to be precise. That's what the theory says.

But the good news is that, if the outcomes are linearly separable, it will have developed a set of rules which will identify all the possible outcomes in finite time.

So, with the bird/plane/glider example you might be willing to give each of the three possibilities to the expert an infinite number of times, but if a good set of rules exists, it will find those rules in finite time. So: there's your answer. It wants 3*infinity examples to work on. But it may well come to a decent conclusion well before it's worked through them all - and it is only with extreme difficulty that one resists suggesting that it might only need half those 3*infinity examples!

At this point there are doubtless those amongst you who will have thrown aside this tract either on the grounds that it's inherently worthless or on the grounds that if you've got a nearly-infinite amount of data collection to do then you'd better get started now.

You are urged, however, to pause a moment. We'll trim the problem down a little.

First, there might, in theory, be an infinite range of different inputs you could provide. But, in practice, you won't have records of all of those possibilities (they'd take up so much room...).

In practice you'll have a nice, finite stack of data for it to start working on and in finite time the performance should become reasonably decent if you just keep on slinging all the examples you have at it.

So what the problem really comes down to is: how to give the expert

plenty of exercise at solving problems without spending the rest of your life at the keyboard and going to bed every night with bleeding fingers.

Fortunately, this all becomes pretty easy. It's just a matter of dull repetition - which is what computers were originally built to do. All you need is to take all of your examples and put them in an array, say E(V+1,N), and write a short routine to let the expert exercise itself in your absence.

The idea is that you set up the examples, get it working over them, and then retire to a local hostelry where you can tell the assembled crowds that your own expert system is currently doing all the work and there's nothing left, therefore, for you to do. That sort of comment always attracts a certain amount of respect amongst the audience because, largely, they won't know whether or not you're telling the truth but, if they think you might be, they will be consumed with powerful emotions.

So, suppressing the urge to retire to the hostelry prior to getting the expert working, we proceed:

First E(V+1,N). Obviously, the N is the number of examples you're going to give the expert. Equally obviously, the first dimension holds the V variables. The V+1 element is used to hold the outcome for that example.

At this point (ie. early on in the process) you could always cheat a bit and, instead of building up this data from scratch, you could fill in the array E at the same time as you input a few items to the expert system in a training session. It doesn't matter much though - it's up to you.

All you need to do now is to put a frequently-repeated loop into the routine so that it keeps on and on working over these examples and altering its rules as it goes. A possible method is to use the index on the loop, say I, to pick which example the expert should work next.

```
10    FOR I = 1 TO N
20    FOR J = 1 TO V
30    V(J) = E(J,I)
40    NEXT : NEXT
```

This does have some disadvantage as the examples are always given in the same order. Maybe this wouldn't matter in some cases. But suppose you had a set of rules which found a correct judgement on example I. Now, these rules might then give a correct judgment on example I+1 so the rules wouldn't be changed. Now suppose we had an example I+2 which the expert got wrong. The rules change and, maybe example I+3 will get the right response but, if example I had been next that might not have

suited the current rules so well - there might have been a mistake and the rules could be altered as a result.

Clearly, to be as foolproof as possible, the expert should try out every possible order of picking the examples and it should work through them time and again. So, if you have N examples, there are N ways of picking the first example, N-1 ways of picking the next example, N-2 ways of picking the next example. In general, there are N! (N-factorial) ways of picking the examples. Where N! = N(N-1)(N-2)...(2)(1).

Which is not only a lot, but it's rather hard to program in Basic.
And, what's more, it should be repeated a few times for good measure.

A more reasonable approach which gives easier code and works for any value of N is to pick the examples at random and just do it a lot of times. Suppose we do it 100 times:

```
50    FOR P = 1 TO 100
60    I = INT ( RND (1) * N + 1)
70    NEXT
```

Doubtless, of course, you're happy about that piece of code - but RND and INT do vary a bit from one machine to another, so let's just define what it does on the Apple:

The function RND(1) gives, with 1 as its variable, a random number greater than or equal to nought and less than 1.

So, multiplying it by N gives a number greater than or equal to nought and less than N.

Adding one to this number gives a number greater than or equal to nought and equal to N or just a bit more. Less than N+1 actually.

The INT function gives an integer number which is less than or equal to its variable - so we now have a number which is greater than or equal to nought and less than or equal to N and which is integer.

So we can use it for picking an example.

This way the expert just keeps on randomly bashing away at the homework you set it and, if you really want to give it a hard time you could always put it into a continuous loop and leave it running until the cows come home.

From our previous in-depth studies the rest of the code is fairly easy:


```
80    FOR P = 1 TO N:D = 0:C = 0
90    I = INT ( RND (1) * N + 1)
100   Q1 = E(V + 1,I): REM :Q1 WAS THE ACTUAL OUTCOME WITH THIS DATA
110   FOR J = 1 TO V
120   D = D + E(J,I) * R(J,Q1)
130   NEXT : REM :THIS FINDS THE VALUE OF THE DECISION RULE WITH THE RIGHT
      OUTCOME
140   FOR K = 1 TO Q
150   D2 = 0
160   IF K < > Q1 THEN : FOR J = 1 TO V:D2 = D2 + E(J,I) * R(J,K): NEXT : REM :THIS
      CHECKS THE VALUE OF THE DECISION RULE WITH THE WRONG OUTCOMES
170   IF D2 > = D THEN : FOR J = 1 TO V:R(J,K) = R(J,K) - E(J,I): NEXT
180   IF D2 > = D AND C = 0 THEN : FOR J = 1 TO V:R(J,Q1) = R(J,Q1) + E(J,I): NEXT :C = 1:
      REM :THIS ADJUSTS THE RULES IF THE CURRENT RULES WOULD HAVE GIVEN THE
      WRONG ANSWER. C IS A FLAG TO MAKE SURE Q1 IS ONLY ADJUSTED ONCE
190   NEXT K: NEXT P
```

Now, of course you can retire to the hostelry of your choosing leaving the expert to do its homework. But take this tract with you (it creates a good impression to be seen able to read a book) for there is a bit more to be said.

Will the expert, when you return, have developed the same set of rules as if you had patiently hammered away at the keyboard for the rest of your life?

You would think so - but it might not. Because the data is being given to the expert in a different way.

Consider. It only alters its rules when it makes a mistake. And it only passed judgement when it was quite sure that no new variables could be entered which would cause it to change its mind, given the minimum and maximum values you so kindly provided.

So, at the keyboard, it might have decided that nothing could change its mind, made a judgement, and been right. Therefore, it would not have changed any rules. Now suppose that two rules, with a given example, gave the same value - a tie occurred. In our previous code it wouldn't have altered the rules unless the outcome it guessed at, from a tie, was the wrong outcome. If it could possibly get away without altering the rules, it did because that saved you from the sweat of giving it any more information.

But, in this case, you aren't having to sweat away at the keyboard so the above code checks out all possibilities and alters the rules in the event of a tie until no further ties occur. So it will, in general, alter its rules more often.

An example:

Suppose you gave it a bird. It checked its current rules and found that it could be a bird but, equally, it could be a glider. But a bird was the first item it came to in the list and, as glider only gives an equal result, it guesses bird. Correct, you tell it. And the rules remain unchanged with bird and glider giving the same values when bird is input. The system still works, though.

But in the code we've just given it the expert won't put up with a set of rules in which bird and glider are the same. It will alter the rules until an absolute difference exists for each and every outcome.

This doesn't really mean that one set of rules is better than another - after all, if it gives the right answers who can argue with the way it does it?

If you wanted the same rules for both cases you'd have to make the code for exercising your expert a bit more complicated. For instance, you could change it so that the rules were only altered:

IF D2 > D, instead of greater than or equal to

or

IF D2 =D AND K >Q

,which means that the rules are changed if a rule is found which gives an alternative, incorrect outcome with an equal score to the correct outcome and this incorrect outcome occurs *before* the correct outcome. It it occurs after the correct outcome it won't cause any problem because, by then, the correct outcome will already have been picked up and another outcome which merely gives an equal score won't displace it.

It's a bit of a fine point but you will notice some practical effects when running your expert system. The most practical effect is simply that it may ask you for more information on the variables before it makes up its mind. This is because it keeps on asking for information until it's certain that nothing can occur to make it change its mind. By making more changes to the rules than before, the actual numbers in the elements of the rule array R(I,J) may have got bigger or smaller. Because of this there will be greater possibilities for variations amongst the various values for the outcomes. So, when the expert is checking your minimum and maximum values on those variables you haven't entered yet, it will seem to the expert that

there's more uncertainty than there was.
And it will go on asking questions of you to try to clear up this uncertainty.

The answer is, if you want to be a purist about it, to use exactly the same methods for exercising the expert as you use for running it normally. This would involve either writing out virtually the same code twice, or mixing in two possible sources of input - from keyboard or the array E(V+1,N) - to the one stretch of code. Either way it's a bit more work and complication and, as it's you that's doing it, it's up to you whether or not you want to give it a try.

5.4 Direct estimation

What we've done so far is to pretend that our expert is going through a training session just as if we were sitting at the keyboard giving it examples one at a time. And there's no reason why we shouldn't continue to think like this. After all, it works.

But the whole gist of the technique is to find a set of rules which will give decent judgements and we might be able to do better than to leave the machine to bash away on a basis inspired mainly by a random number generator.

When we were considering the case of only two outcomes we pointed out that it would be possible to adjust the rules after every example whether the current rules gave the correct answer or not and we could do the same with more than two outcomes.

In this case we'd ignore the random element and work through all of the examples we provided to the system as follows:

```
10    FOR K = 1 TO N assuming that we have N examples
20    FOR I= 1 TO Q working through all of the possible outcomes
30    FOR J = 1 TO V working through all the variables

40    IF I = E(V + 1,K) THEN :R(J,I) = R(J,I) + E(J,K)
50    IF I < > E(V + 1,K) THEN :R(J,I) = R(J,I) - E(J,K)
60    NEXT : NEXT : NEXT
```

What this does is to add the example values to the rule which gives the correct outcome and subtract the example values from all of the rules

which belong to other outcomes. No reference is made to the question of whether or not the current rules would have worked or not.

At the end, after working through all of the examples in this way, we're left with a set of rules whose values (in theory) we know quite precisely.

Suppose, for example, that we have three outcomes. Think of them as the corners of a triangle hanging in space somewhere as in Fig. 5.1. The rules consist of vectors - lines in particular directions - which each point to one corner of the triangle and away from the other two corners.

The corners of the triangle are, in fact, determined by the average values of the variables for each of the outcomes. Suppose that outcome 1 has average value x_1 for some variable or other and outcome 2 has average value of x_2 and outcome 3 has average value x_3. Then the rule vector for outcome 1 contains $x_1 - x_2 - x_3$.

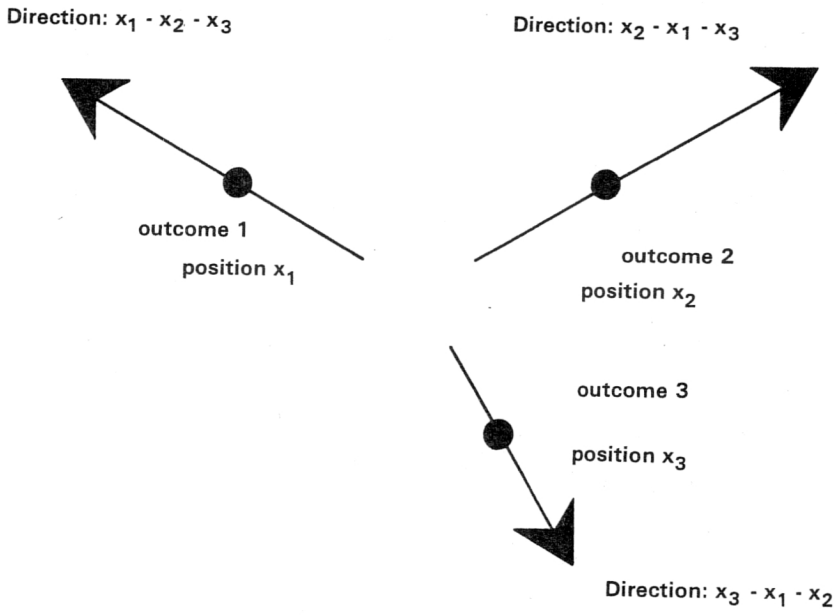


Fig. 5.1
Direction vectors for 3 outcomes

Likewise, the rule vector for outcome 2 contains $x_2 - x_1 - x_3$, and so on.

Each rule vector tries to steer a course as much in the direction of its own average value as possible - but, by subtracting the average values of the other outcomes, it also tries to steer away from them as much as possible.

You might have noticed that we haven't really used average values. Say, there were n_1 examples of outcome 1. We haven't divided x_1 by n_1 to get the true average.

First consider what would happen if there were an equal number of examples of each outcome. Well, it wouldn't make any difference if we divided by the number of examples or not. Only their relative size is important so nothing would have been gained or lost.

Now suppose that there really are a different number of examples of each sort. Well, it could be that this is important. If you really give the expert a representative set of examples and that set of examples contains, say, one outcome which occurs twice as often as another outcome then this is useful information. The knowledge that one outcome is twice as likely as another and the fact that there are twice as many examples of it might cause you (or the expert) to think that twice as much faith (to use a scientific term!) should be placed in the outcome with the most examples. So, we might, having got the mean values for that outcome, 'weight' them by multiplying them by two.

In general, if there are n_1 examples of x_1 we might multiply x_1 by n_1. And, if x_1 had been the average value of the n_1 examples we'd have been back where we started. Simply adding up all of the examples would have been quite adequate.

The snag is that things might not always be as simple as they seem. If we were still trying to predict the difference between a bird and a plane and we gave the expert one example of each it could then go away and work out a set of rules. But suppose, for some reason, that you give the expert two examples of the same thing. What would this mean?

The same set of rules would still separate the different possibilities quite nicely and there's no reason to think that one outcome is twice as likely as any other. In essence, what you give it might not have been truly representative at all. And that would cause a bit of a problem. It's really up to you to judge what you should do here.

For a start, you could assume that the values you provide are fairly representative and work from the average for each outcome. After that, you could consider the question separately as to whether the number of examples in each outcome is important or not. If it is, you could weight the averages by the number of examples with each outcome and possibly gain some benefit. But if you feel that the number of examples with each outcome isn't really a helpful guide - possibly because you just dreamed them up on the spur of the moment - then you could stick with the ordinary unweighted averages.

Clearly, to get the averages you need to keep count of how many examples you've got of each possible outcome. You could do this by splashing out on another array, DIM N(Q) to keep track of it.

So:

```
FOR I=1 TO N
N(E(V+1,I)) = N(E(V+ 1,I))+1
NEXT
```

And, then, divide the R(I,J) by the values in N(J) to give mean values.

The only snag with this method is that, in all honesty, it doesn't always work. Consider the following values for V(I):

	Bird	Plane	Glider
Wings	1	1	1
Beak	1	0	0
Engine	0	1	0

So a bird has wings and a beak, a plane has wings and an engine, and a glider has wings and very little else. Now work out a set of rules R(I,J) using average values and their differences:

	Bird	Plane	Glider
Wings	-1	-1	-1
Beak	1	-1	-1
Engine	-1	+1	-1

Finally, give each of the sets of variables, in turn, to these rules to find the values for D(J) with each V(I) input to R(I,J):

Examples D(J) for each possible outcome:

	Bird	Plane	Glider
Bird	0	-2	-2
Plane	-2	0	-2
Glider	-1	-1	-1

Clearly, it hasn't got a clue about the glider, even though it can guess the other outcomes with no trouble.

This problem wouldn't have arisen if Glider had been possessed of some other variable which uniquely identified it - like Towrope, for example. But, as it is, it hasn't and the method has broken down. It can easily be put right by counting Glider twice so that against Wings we have -2 for Bird and Plane and 0 for Glider - but we didn't know that until we checked out the rules to see if they worked.

So, although it might seem useful to attempt a direct estimation of the rules, it's always a good idea to check them out afterwards. Maybe just by running the expert on a few examples. But, while you're doing that it might as well be building up its own rules in its own way - which rather takes us back to exercising the expert, just as we were doing before.

Chapter 6

Running for Real.

6.1 Using your expert.

By now you've got, believe it or not, the makings of a general-purpose, fully-fledged, expert system with which you can amaze your friends and colleagues. You've been through all the important stages and, in case you hadn't realised it, they were these:

You've given the system the names of all of the possible variables and you've told it what you think are the minimum and maximum values for each variable. The variables can be intrinsically numeric or just categories, like Yes/No data. Either way,it's treated as numeric input.

You've given the system the names of all the possible outcomes. As the expert is going to choose one of these outcomes every time it's asked for an opinion you'd better make sure that at least one outcome in the list will always be applicable. Ideally they should be mutally exclusive and the sum of the probabilities of each occuring should be one. Which means that only one outcome can occur but that there must always be one outcome which actually does occur.

You've given the system some examples (as many as possible) of likely inputs and told it what the outcome was for each input.

You've set up a training session in which the expert keeps on churning around the examples you've given it and modifying its set of rules until it's got a set of rules which always gives the right answers or, at least, until it's absolutely sick of trying.

But now, you actually want to use the expert for something. Obviously, you can carry on just the same as you would do for a training session. Say, you've set it up to predict the weather.

The day dawns, you go to your expert, key in a few variables, and it gives you a forecast. No trouble. And then, the next day, you do the same. And the next day. But suppose the forecast is wrong ? Well, if it's running as a training session, the first thing the expert would want to know is if its previous forecast was correct so that it could adjust its rules if it wasn't. Now, in this case, there's no particular hardship in satisfying its curiosity on the matter. You could let it know the outcome for the last set of data you gave it and then ask it about a weather prediction for today.

But, in your haste to pack umbrella, raincoat, wellington boots and all the usual equipment needed for an English summer you might either forget to let the expert know what happened to its last forecast or you might just want to forget.

Likewise, you might have set up the expert to carry out far-reaching medical diagnosis. You've had a training session and there you are, sitting in your consulting room with a queue of patients outside the door.

The first patient enters. You switch on the machine.

Typing his symptoms into the expert the screen flashes up the chosen outcome.

"Sorry, old chap," you observe sagely, "not your day, by the look of it. Only seven days to live. Next !"
And, at this point, you have to sit around with the next patient for a whole week before you can let the expert know if its predicted outcome was correct for the first patient. It's the sort of thing that loses you customers if you don't make provision for dealing with the situation in advance. And besides, you want people to get out of your consulting room as fast as possible if there's any chance that they may be carrying germs.

The point is that, overall, although it's a good idea to give the expert as much continuing feedback as possible there comes a time when you want to stop training and simply get on with the business of making expert judgements. And, in programming terms, all that this involves is going through the training routine but assuming that every response from the expert is correct. It doesn't ask you any more if it was right or not and it doesn't adjust its rules either.

In a nice clear-cut example, like the bird/plane/glider, when the expert can quickly get to be right every time - it makes little difference if you stop giving it any feedback. But in more usual cases - like weather forecasting - when you know, or are pretty sure, that mistakes will still occur from time to time the expert will lose out a bit by missing its feedback. You can

largely get around this though by storing the details of the examples that occur and filling in the outcomes later when you feel like it - or when the outcomes become known.

That way you gradually build it an ever-bigger set of actual data which you can periodically use to exercise the expert to make sure that it isn't getting out of touch with its subject.

If, now, we go back to our weather example you'll see that we get different results when we run it for real to the results we got on a training session. During training, everytime a mistake was made the rules were adjusted. Now, we have a replay of March 1982 using the heavily-exercised, rules without change throughout. The results are in Table 6.1

At first glance we might think that the results aren't wildly impressive - after all, it has made nine mistakes during March. But if we go on to apply the same rules, without any feedback, to April (Fig 6.2) we find that we only get six errors in 29 days. So, over these two months we've got 15 errors in 60 days - a 75 per cent success rate. Which, again isn't perfect, but it's far better than chance. After all, that sort of success rate suggests that, if you were actually using expert, it would only be wrong 1.75 days of the week!

And, as an added bonus, you could consider that it doesn't matter much if the expert predicts rain and the rain doesn't arrive. The worst that happens is that you walk around with an umbrella you don't need. The bad news is when expert says it's going to be dry and it rains. And this happened on only four days in March and April 1982 - about every other week. So maybe the news isn't all bad.

The interesting point to note about April's weather though is that it wasn't this weather on which the expert was trained. Suppose that weather prediction was a problem that really was linearly separable. It isn't, and that's why it's such a hard test for our expert system. But suppose it was, and expert was trained on March's weather. Then, after due exercise, the expert could infallibly predict the weather for March. But that doesn't mean that it could predict the weather for April - which might have entirely different values. The fact that it has produced a fairly decent set of predictions is something of a bonus inasmuch as it suggests that there is something about weather forecasting in general which our expert system can do for us. It can take genuinely novel data and come up with something like a reasonable solution even though we don't know ourselves exactly how to predict tomorrow's weather and have only been able to give our expert system the most general of approaches to work with and a few examples to help get it going.

DAY OF MONTH	ACTUAL WEATHER NEXT DAY	WEATHER FORECAST BY EXPERT
1	Rain	Rain
2	Rain	Rain
3	Rain	Rain
4	Dry	Rain - Error
5	Dry	Rain - Error
6	Dry	Rain - Error
7	Rain	Rain
8	Rain	Rain
9	Rain	Rain
10	Rain	Rain
11	Rain	Rain
12	Rain	Rain
13	Dry	Rain - Error
14	Rain	Dry - Error
15	Rain	Rain
16	Rain	Rain
17	Dry	Rain - Error
18	Rain	Rain
19	Rain	Rain
20	Rain	Rain
21	Dry	Rain - Error
22	Dry	Dry
23	Dry	Dry
24	Dry	Dry
25	Dry	Dry
26	Dry	Dry
27	Dry	Dry
28	Rain	Dry - Error
29	Dry	Rain - Error
30	Rain	Rain
31	Rain	Rain

Table 6.1
Weather forecasts for March 1982, London

DAY OF MONTH	ACTUAL WEATHER NEXT DAY	WEATHER FORECAST BY EXPERT
1	Rain	Rain
2	Dry	Dry
3	Dry	Dry
4	Rain	Dry - Error
5	Rain	Rain
6	Rain	Rain
7	Rain	Rain
8	Rain	Rain
9	Dry	Rain - Error
10	Dry	Dry
11	Dry	Rain - Error
12	Dry	Rain - Error
13	Dry	Rain - Error
14	Dry	Dry
15	Dry	Dry
16	Dry	Dry
17	Dry	Dry
18	Dry	Dry
19	Dry	Dry
20	Dry	Dry
21	Dry	Dry
22	Dry	Dry
23	Dry	Dry
24	Dry	Dry
25	Dry	Dry
26	Dry	Dry
27	Dry	Dry
28	Rain	Dry - Error
29	Dry	Dry

Table 6.2
Weather forecasts for April 1982, London

6.2 Reserved judgement

If, when attempting to make a decision, our expert system couldn't make up its mind what the outcome might be the best thing it could do would be to say so. Admit defeat rather than putting out a simple guess.

Unfortunately, this is a particularly difficult thing for it to do.

Suppose it was back with the bird/plane/glider example and, early in its training, it found that all three outcomes gave identical values on a given input. As the expert is built so far it would make a guess as to the correct outcome if this happened, and if it was wrong, it would correct its rules. But, you say, if all the rules give identical answers it could reply: Don't know. Rather than pretending it did know when it really didn't. Yes, fine. Of course you could program that in. A slight alteration to the code and, if there's more than one most likely outcome, print "Don't know" or, better, list the several possible outcomes. That would tell you a lot more about the judgement that had taken place and, so, you might as well include it in the program now.

But will it ever use this extra bit of code? Yes, sometimes. For instance, with that bird/plane/glider example it may well give a Don't Know - reserved judgement - answer while it's training. But, once its rules are established it won't reserve judgement at all because it will always get the answer right. Which is just as you'd want it to do.

But now turn your attention to the weather (again).

What do you think the odds are of a given set of inputs concerning the weather giving rise to two or more identical judgements on the basis of a given set of rules? It's hard to calculate an exact answer to that question: but, certainly, it's approximately nil.

The variable values you input are continuously variable. REAL variables they can vary infinitely (within the limits of the machine) from any one value to any other. Take several of these variables and combine them with a given rule which is also made up of REAL values. Take another, different rule with REAL variables and see if you get the same answer. In general you won't. Even if two answers are very similar it's going to be most unusual for them ever to be exactly the same.

Further, if you think back to the rules as a method of placing a surface between the different outcomes, you'll see that as long as the surface does separate the outcomes it doesn't matter exactly where it is. Likewise, it doesn't matter exactly what values are in the various rules as long as they

work. So those values could be altered quite a bit, thereby altering any equalities which might have occurred, without upsetting the main working of the expert.

Naturally, you'll say that you don't need the values to be exactly the same in order for the expert to reserve judgement - just similar. But how similar is similar? And to a machine? If you divide all of your variables by, say, 100 before applying the rules the differences between the values given by each rule will be reduced by a factor of 100. So: they're more alike, at least they give more similar answers and so, surely, the expert could reserve judgement more readily. Obviously this isn't right. But what would be a correct way of working?

As we've been thinking in terms of categories so far, we could introduce an extra outcome and call it: Don't know. But how would you train the expert on that outcome? The whole point of the training sessions is that you certainly do know the outcomes. If the expert asks you if Don't Know is the correct outcome, how would you answer? Perhaps you should give up and reply that you don't know either.

The problem in essence, is not one of deciding whether or not the expert really does know the answer or not. It's the problem of deciding the extent to which it believes in each of the possible outcomes individually. Its final answer will be to suggest that outcome which it believes in the most. But, if this is very similar to the extent to which it believes in some other outcome, then you'd like it to reserve judgement, or at least tell you about it all.

And, with the expert system we've built, this isn't really possible. Our expert system is designed to accept any input on any problem and give the best answer. If you want to know how much confidence you can place in that answer there is only one way to proceed. That is to give it some data and see how often it gets it right.

For the bird/plane/glider example we found that, after training, it got the right answer every time. So its opinion is expressed with probability 1. For the weather forecasts it was right 75 per cent of the time. So its answer is expressed with probability .75. But these are figures for the answers in general. They are not figures for isolated, specific answers. One answer may be given with well-nigh absolute certainty and another with a great deal of uncertainty. But the problem is that we do not know which is which. Nor, for this general case, do we have any means of finding out.

6.3 The problem of distance

Now, surely, one might think that we could work as follows: We have a decent set of rules in our expert, built up after a longish training session. We then give the expert another item to decide on. It could then list all of the possible outcomes and, against each, list the values which the rules gave for this outcome with the given input. The expert will pick that outcome with the highest value and, surely, this value, and the values for the non-chosen outcomes, will reflect the probabilities for each outcome being correct?

Sadly, no. They won't. The expert has been trained to pick the largest value but it has been told nothing about what these values actually mean in terms of probabilities. The biggest value denotes the most probable outcome - but how much more probable is, to the expert, a complete mystery.

It's all tied in with the problem of distance and distance measures - and that's what we'll look at now.

In the current, learning, system the expert has to place a boundary fence between several different outcomes. As long as the fence accurately separates the outcomes it doesn't matter exactly where it is. What matters is simply which side of the fence the variables fall - it doesn't matter at all how close to the fence they are. And in this sense, the concept of distance doesn't occur to it.

To help get these ideas straight let's imagine that we have before us a map of a strange country. As you can see in Fig 6.1. It has two axes (North-South and East-West) and it has two locations marked on it (X and Y, say) and it has contour markings to denote the height of the terrain. Locations X and Y are, we see from the contour markings, each situated on the top of a mountain. (refer to fig 6.1.)

So, we now have Mountain X and Mountain Y and, we see on the map, that point P is, more or less, on Mountain Y, which means that anyone could look at the map and say which Mountain point P belonged to.

Having set up this map, we can now look at the various ways in which are expert could solve the problem.

Using the learning algorithm all the program does it to lay down a boundary fence between the two mountains and as long as point P is on the Mountain Y side of the fence then the boundary is in the right place

because point P will be classified correctly. But that doesn't really tell us to what extent P belongs to Mountain Y - it could be right on the peak, at point Y itself, or right down in the foothills. At best, all it really tells us is that point P isn't part of Mountain X.

So now to measuring distances.

The easiest way, and the one that would first occur to you, is to measure the distances from P to X and from P to Y - and classify P according to which is the least distance. That's not a bad method to use so we'll do it.

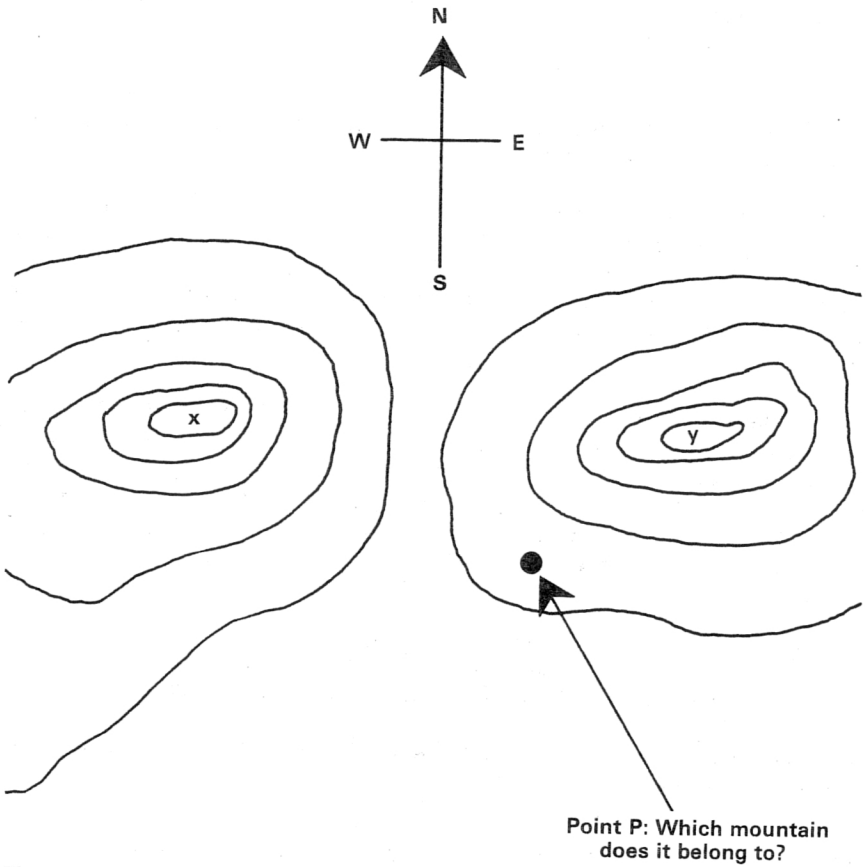


Fig. 6.1
A physical analogy for distance measurements

Unfortunately, it isn't just a question of getting out a ruler because what the computer has is not a map, as such, but a series of map references denoting measures along the two axes. So point X has co-ordinates in North-South and East-West as (NS_x, EW_x) and point Y is at (NS_y, EW_y) and P is at (NS_p, EW_p).

To calculate the distance of P from say X we calculate: $d^2 = (NS_x - NS_p)^2 + (EW_x - EW_p)^2$ which is the Euclidean Distance measure (squared to keep everything positive) of P from X. We calculate the same thing for distance P to Y and we've then implemented a Minimum Euclidean Distance Classifier for point P.

The fact that the mountain example works in real space and our problems exist in an n-dimensional description space makes no difference to the way we now proceed. All we need to do is to work out the 'position' of each variable for each of the categories denoting an outcome. Taking the bird/plane example with two variables, beak and engine, we have, for a bird, co-ordinates (1,0) - beak and no engine - and for a plane (0,1) - no beak and an engine. Given then an object P which has co-ordinates (1,0) we can classify it as a bird because this gives the minimum Euclidean Distance measure out of the two possibilities. In fact, if in array R we held the "positions" of each outcome we could input a new object in array V and calculate:

```
FOR I= TO V
D = D+(V(I)-R(I,J))↑2
NEXT
```

and choose that outcome J which gave the minimum value for D. No trouble. Working like this would, in fact, be quicker than implementing the learning algorithm.

The point to be made now is: that, maybe, without noticing it, we actually were very woolly about what we actually asked the expert to do. When we asked to which category point P belonged we didn't really define the categories at all well because we could have been asking either : which is the shortest distance , P-X or P-Y? Or, to which mountain, X or Y, does point P belong? And, whereas the distinction doesn't really matter too much in the bird/plane example, it would matter a great deal, for instance, in weather forecasting.

Staying with the bird/plane example, all we did was to find the minimum distance of P-X, P-Y, and carry out our calculations more or less as the crow flies.

We were able to do this because the variables were point functions - they only exist at a single point and, if you like, the nature of the terrain between the points is quite immaterial.

If we'd been working with continuous variables, rainfall figures say, then we wouldn't have had point functions we'd have had continuous variables and these would define an intervening terrain. What is more they would have needed a bit of calculation in order simply to ascertain where the mountain tops actually were.

In general, the position of the mountain tops is best calculated by finding the mean (or average) value of the variables for each outcome. If we have n observations for a given outcome then the mean, m, is:

$m = (x_1 + x_2 + x_3 + ...x_n)/n$ which is all fairly familiar.

So, whether we've got discrete or continuous variables, we could still proceed by calculating the mean values for each outcome, placing them in array R, and then implementing a Minimum Euclidean Distance Classifier.

The problem still remains though that we've calculated the distance to the mountain tops - we haven't fully attempted to answer the question of which mountain point P belongs to.

To see why this might matter, suppose that the mountains were of different sizes - one might be a proper mountain and the other much more like a molehill. Now suppose that point P is only three feet away from the centre of the molehill and five hundred feet away from the centre of the mountain. Now: does point P belong to the mountain or the molehill? Using the minimum Euclidean Distance Classifier it certainly belongs, not to the mountain, but to the molehill. But is this right? After all, three feet away from a molehill is a long way - well clear of it. Whereas five hundred feet from the centre of the mountain might be half way up it. You see the problem.

That's where the contour lines come in. If we were looking at a real map we'd intuitively take the size of the mountains into account in answering the question - and that's what we must try to do with the problem of classifying objects now.

In terms of variables the contour lines represent the probability distribution function (pdf) of the variable - the normal pdf is just one example, and, viewed from the 'side' it looks like this:

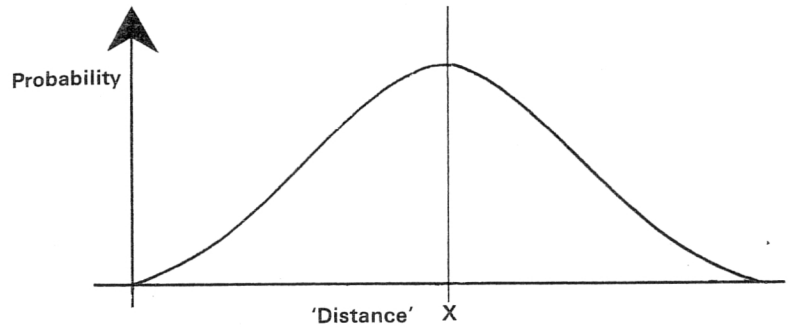


Fig. 6.2
The normal probability distribution function

Point X is the mean value for variables with that pdf and, if it were measured in two dimensions and viewed from the 'top' it would look just like the picture of a mountain with contour lines.

Now, associated with every pdf is a 'standard deviation', sd, which is the square root of its variance. The standard deviation is:

$$sd = \sqrt{((X_1 - m)^2 + (x_2 - m)^2 + (x_3 - m)^2 + ...(x_n - m)^2)/n}$$

So you calculate the mean, m, first for the n observations on each variable and then calculate the sd.

The sd is a measure of the 'size' of our mountain - it measures the spread of each variable. So we can make all of the mountains the same size by dividing all of the measurements by the sd. This is called Scaling to Unit Variance.

Working from a sample set of data it's possible to calculate the means, calculate the sd, then calculate a new 'distance' measure as:

$d^2 = ((x-m)/sd)^2$

which gives a minimum distance classifier scaled to unit variance.

This will certainly help the expert in making decisions - but will it enable the expert to say with what probability its statements are true?

Well, the answer is: maybe.

Suppose that the variables were, actually, normally distributed (that is, they have the normal pdf). The measure $d=\sqrt{d^2}$ has been tabulated for the normal pdf and it is possible to look in the tables (or calculate your own tables on the computer - not a very easy task) to see just what the probability is that, with a given value of d, the new observation belongs to mountain X (or, in statistical terms, to see what is the probability that the given observation comes from a given population). Doing this for each outcome would enable the expert system to make statements with probabilities attached to them.

The problem that really arises though is that many variables aren't normally distributed and it can be quite difficult to determine exactly what their pdf is.

Returning to the bird/plane example with two variables Beak and Engine, it is immediately obvious that these variables aren't normally distributed. As point functions they don't resemble mountains so much as lamp posts - with everything concentrated in one place. In fact, that makes life somewhat easier, in this instance, because we don't need to use tables - the probability is either 0 or 1 depending on which variables and which outcomes we have so the system can make exact probability statements about its opinions. The 'distance' measure will always be either 0 or greater than 0 and the probability for each classification being correct will be 1 if the distance is 0 and 0 if the distance is greater than 0. Easy.

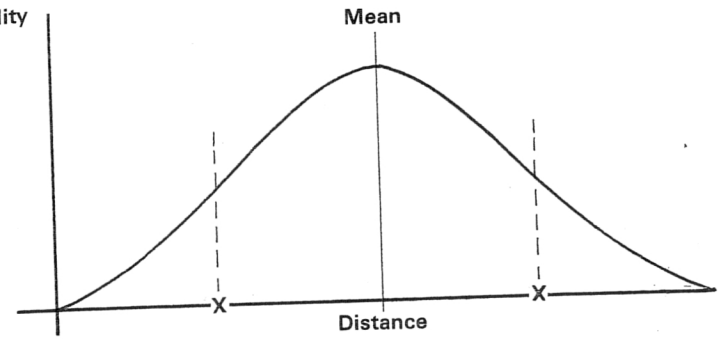
The real problem occurs when we simply don't know what the pdf is for a given variable - and this might be the case for, say, rainfall figures. We can be pretty certain that it isn't normally distributed because it isn't symmetrical about the mean value. And we know it isn't a point function because we have continuous variables. So: what is it? And, if we knew the answer and implemented it on our expert we'd then have an expert system which was very good at weather forecasting but, probably, quite useless at anything else with a different pdf. (see fig. 6.3)

The concept of distance measures is a good one - but translating these distances into probability statements can be a very complex task, because distance and probability are not the same thing at all.

The problem of distance is really a very general one and shades into the problem of similarity.

Suppose that we had a given set of variables and we wanted to decide an outcome. What we might do is to run through the set of values the expert was trained on to see how many of these items were exactly the same as the values we've just got. We could then get the expert to pass judgement

a) For a nicely-behaved pdf, like the normal pdf, probabilities vary in a very regular fashion as the distance from the mean varies.



b) For less well-behaved functions the relationship can be much harder to define. In this "skew-left" example below-average readings can be much more likely than some above-average readings.

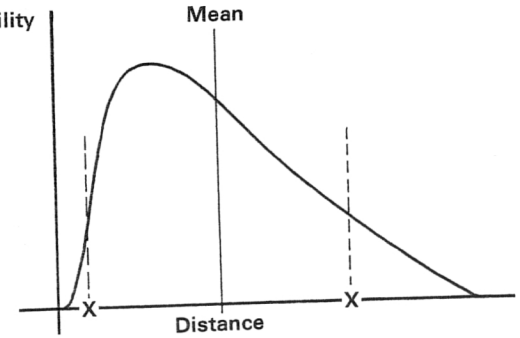


Fig. 6.3
Symmetric and asymmetric probability distribution functions.

by choosing that outcome which contained the largest number of examples with exactly the same variable values. And this isn't a bad way of working if the data suits it.

It has the value, in its simplest form, of not relying on the distance measure - so we can use categories of items or any other method of describing the data we like. For the bird/plane/glider example it would work well.

But, back to the weather. In this case the chance of finding even one set of identical readings is very remote - so how do we measure the similarity? Back to the distance measure and all of its problems. It's a bit of a pity really, because, if it worked we could actually assign probabilities to the various outcomes based on the number of similar cases which fell into each category. Certainly, if you had a specific problem which suits this method of working you'd find it simple to use, easy to understand, and reasonably accurate. The sort of situation in which it would work might be the case in which your variables consist of a number of categories but these categories don't precisely define any particular outcome. In the bird/plane/glider example the categories did exactly define the outcome. If, though, you had a list of medical symptoms which indicated a particular complaint but did not point with absolute certainty to exactly which complaint then this method would give as good results as any. It would, in fact, give as good results as our original method (which is one reason for sticking to that method) but it would also be able to indicate the extent to which it thought its judgement might be correct.

A variant on this technique is, obviously, to use the original method to make a judgement and then to search through past records to check for exact instances which might be identical so that an indication might be given of other judgements that might be considered.

6.4 Understanding your problem

The whole essence of this approach to building an expert system has been to produce a method which will work with a reasonable measure of reliability in any circumstances. It must produce a moderately decent outcome irrespective of the problem you give it and, more to the point, irrespective of whether or not you actually understand the problem.

Take weather forecasting. You may be working at the Meteorological Office, but you probably aren't and probably don't know how to predict the weather. But you can still develop an expert system to help you to do it.

The same goes for many other problems you might think of. You can use the same basic expert system to have a crack at a wide range of problems.

But, if you really have a problem which you understand well, then you'll stand a chance of getting much better more precise results if you make up a tailor-made program for it. The point is: if you really understand your problem you won't need anyone else to tell you how to do it. Will you?

Chapter 7

An Expert on Everything in the Entire Known World.

7.1 Nodes

Having got your expert system up to this stage there's one very special application for it. Your can get it to become expert in absolutely Everything in the Entire Known World. It sounds difficult, perhaps, but it isn't really. Just a bit time-consuming.

First, set up the system as before with seven variables and five outcomes. Say:

Variables	*Outcomes*
Wings	Bird
Beak	Plane
Engine	Glider
Min. Temp.	Rain tomorrow
Max. Temp.	Dry tomorrow
Rainfall	
Sunshine	

Now you start running a training session. First, teach it about the secrets of birds, planes and gliders by providing examples of things with or without wings, beaks and engines. If it asks about the other variables reply with a zero or something equally non-commital. When it seems to have got the hang of that, teach it about the weather and, if it asks about wings, beaks or engines, give it zeroes again. Now let it run away and practice for a while.

Now, believe it or not, you will have an expert system that is simultaneously expert in predicting the weather and in identifying flying objects.

The extension of the situation to cover every field of Human Endeavour is obvious. You just keep on adding new variables and new outcomes and new training examples. And the expert will, finally, become expert in everything. Subject, of course, to the limitations of your computer's memory size.

"But surely," you exclaim, "this is wonderful!"

Well, yes, it is, of course. One doesn't waste one's time designing trifling systems when there are the entire problems of the world to be solved. But you will, possibly, find it a little slow. And you may, possibly, get irritated if it keeps on asking you about beaks and engines when you really only want a weather forecast. For one of the difficulties in this scheme is that, as it stands, the expert is not only capable of solving every problem in the Entire Known World. It will actually, really, try to solve every such problem every time you switch the wretched thing on. It's a noble attempt on its part, of course. But the admiration one feels for it can soon pall.

The answer, obviously, is to have two versions of the expert. One which is good with flying objects and one which is good at predicting rain. And further versions which are good at other things too, if you like. And you don't connect them to each other because there's no point in connecting them.

Which brings us to nodes.

For what we've designed is a node. There is one node for flying object identification another node for weather forecasting. The two nodes aren't connected because they have no need to be. But each node has several inputs (variables) and several outputs (outcomes) and they could have been connected if we'd wanted to connect them.

For instance, suppose we had two expert weather forecasting nodes. We could have connected the output from one node to one of the inputs on the other node. And, then, we could have used the first node for predicting, say this afternoon's weather and the second node for predicting tomorrow's weather.

Then, when the expert had formed a prediction for this afternoon it could use that prediction as an input to its long-range (tomorrow!) forecast.

However, going back to the one big node which was expert in absolutely everything, that big node was simply the ultimate example of a parallel process. You told it everything it wanted to know and it told you something in return. By splitting it up into two smaller nodes - weather forecasting and flying object identification, say, - we changed it into two parallel processes.

As they won't be executed simultaneously, they are two parallel processes arranged sequentially. As they don't have anything to do with each other they aren't connected. But, if they had been connected we would have had a kind of sequential process built up out of a series of nodes (two, in this case) with a tree structure, if you want to think of it like that.

Each node could be set up and trained individually so it would be just as good as if it was the only node in town.

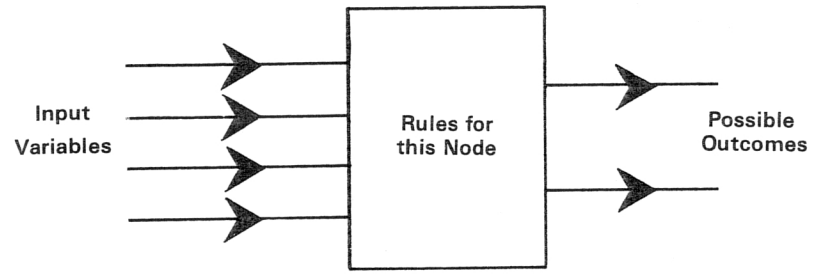


Fig. 7.1
A single node.

And, although in theory it's no better than one big parallel process, in practice it might help save you some time, as well as giving you a bit more information about what's going on.

Think, for another boring instance, about the weather again.

Currently, you have a node which will tell you if it's going to rain tomorrow or not. Fine. But it could have told you other things. Like, for instance, whether or not it was warm today. Now surely, one comments, any idiot can work that one out? True, but this is a computer so it's a bit more complicated than that.

Set up another node and start training it. Get it to say Warm Today if the temperature is over, say, ten degrees Centigrade. It should get the hang of that without too much trouble even if you, personally, have some doubts about ten degrees Centigrade being warm.

Now go back to your first node and give it five variables - max. temp., min. temp., rainfall, sunshine, and warm today. The warm today is the output

from the first node and is obviously 1 if the max. temp. is over ten degrees Centigrade and 0 if it's not. Train the second node to predict rain or dry tomorrow. The process is shown in Fig. 7.2.

Now, you should notice that, for some strange reason, it behaves differently to the way it behaved before with only four variables.

It's almost as if it had some extra information - but all you've given it is a variation on the exact temperature which is something it already had.

Not quite. You've given it some extra information. That item about warmth is extra to what it had before. Not very much extra, but a bit. The point is that Warmth, as we've just defined it, is a non-linear transformation of the maximum temperature. As the expert only works with linear transformations you've given it something it didn't have before and that it couldn't ever have worked out for itself.

One way of thinking about it, if you're not convinced, is to suppose the expert only had the four variables minimum temperature, maximum temperature, rainfall and sunshine. From these it works out a set of rules for predicting the weather. You then ask for a forecast and,ignoring for the moment the other three variables, it picks out maximum temperature and multiplies that by a number in its set of rules. According to the answer it makes its prediction.

Now add in Warmth. This is 0 if the maximum temperature is below ten degrees Centigrade. So, below that temperature, everything is calculated as before.

But as soon as the maximum temperature goes above ten degrees Centigrade Warmth gets the value 1 and this is multiplied by some value in the new set of rules and added to the final results. So, suddenly, as the maximum temperature rises the decision process is given a boost, as it were. Mathematically, this is called a step function and it is non-linear.

So now you have an extra variable,a real, new, extra variable and, obviously, you could just say that you've got five variables and, every time the expert asks if it's warm today you could just look at the maximum temperature and give it a 0 or 1 accordingly.

But you could also get the expert to do this for you by setting up a special node for it. This node only really needs one variable for input - maximum temperature - and you could go to that first, train it to recognise warmth, and get the answer to that matter first. Then you could take the outcome - warmth or not - and use it as the input to the next node, which is the one

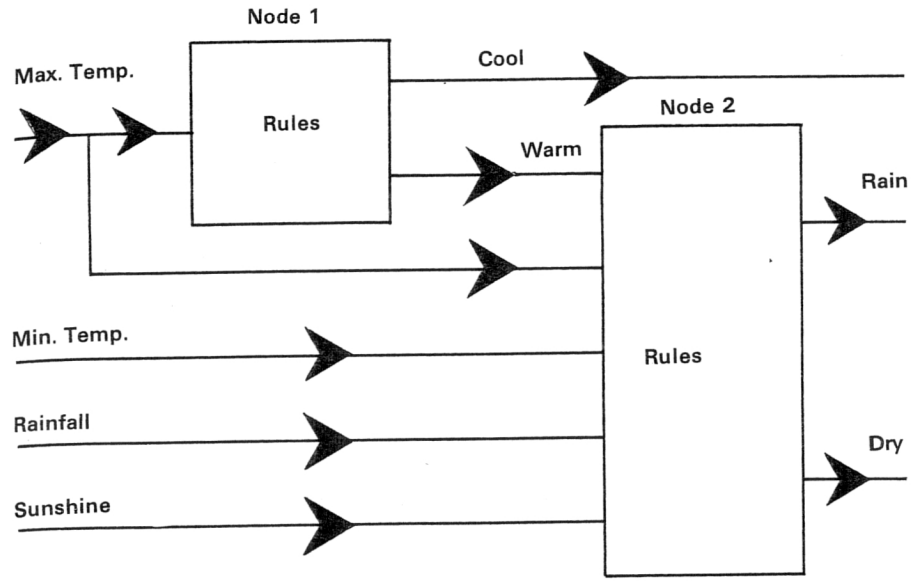


Fig. 7.2
Adding an extra node.

Several nodes can be combined together to form a network giving intermediate conclusions and inputs to other nodes.

that predicts rain. It's a bit like messing around with the wiring on a hi-fi set. You take the output lead from one unit and plug it into another unit. And maximum temperature is an item with leads which go to both units. And, like a hi-fi set, you can keep on adding fresh leads and plugging bits in and unplugging them until you go blue in the face. Hopefully, it will start working before the strain destroys you.

It takes a bit of alteration to your program to do this - but not too much. Suppose you decide that your program has N nodes. Then, when you set up your variables and rules, you could write DIM V$(V,N), Q$(Q,N), R(V,Q,N), V(V,N)

In other words, just add another dimension to every variable - most of which are arrays anyway, and then reference each node in turn simply by specifying N.

And, say you've come to a node which makes a judgement Q$(J,N1) - outcome J on node N1 - a short routine can search all the other variables to see if this outcome is an input variable for any other node anywhere else. If it is then V(I,N2)=1 for variable I on node N2 assuming that V$(I,N2)=Q$(J,N1).

It's a bit like automatically scanning around to see if any switches can be set in the system.

And, having set a switch, this is equivalent to providing the expert with another variable value. So, of it's own volition, it can now check around to see if it can draw any further conclusions. If it can, then that's another outcome it can work on for possible inputs to new nodes. And so on, and so forth, until it comes up with something you think is actually useful. Like the answer to the original problem you set it.

Of course, the real difference in the expert's behaviour with more than one node lies in the fact that, typically, at each node non-linear transformations of the data take place - it isn't the presence of nodes as such that make it act differently.

But there is an incidental advantage. The nodes give you an idea of what's going on in your expert's head. For each of these nodes provides an outcome which you might think of as an intermediate solution. Like:
Input: Maximum temperature today
Output: It's warm today!
Input: Minimum temperature, Rainfall, Sunshine
Output: I bet it's going to rain tomorrow (gloom descends).

Exactly how your expert replies is up to your programming - but that's roughly what it can do and, put like that, the matter of whether or not today's weather is warm doesn't sound quite so trivial. After all, it might almost be people talking if, of course, people were actually in the habit of flashing their words up on a screen everytime the weather was mentioned.

Doubtless any application of your own will have lots of nodes and lots of variables and lots of outcomes and deal with something much more non-trivial like, for instance, DIY Brain Surgery. If this is the case, having the Expert show you a few intermediate conclusions before it finally advises you "Remove patients head" could help avoid much acrimony.

7.2 The variables so far

By now, unless you've been concentrating hard and taking notes at the same time, you've probably got an amazing collection of unrelated statements lumped together under the heading Expert System. It really is quite surprising how these things can creep up on you. All you do is write a bit of code, add another bit, modify the first bit, add a bit more, think of something else, wander off to make some coffee, have a bright idea, and add that . . .

What you really need is a bit of a sort out. So, here it is - a collection of all the main variables you might have used so far.

```
DIM V$(MV,N)
```

This is the list of all the variable names that you have in your expert system. There is one list for each node. The maximum number of variables you can have at any one node is MV and the number of nodes is N.

When you're first setting up the expert and want to start DIMensioning arrays the first thing you should get the system to ask is:

```
INPUT "HOW MANY NODES HAVE YOU?"; N
```

after which it can ask:

```
INPUT "WHAT IS THE MAXIMUM NUMBER OF VARIABLES YOU HAVE AT ANY ONE NODE?";MV
```

- or words to that effect.

Obviously, you're going to waste array space doing things this way, unless all of the nodes happen to have the same number of variables going into them. But, although this can be avoided, it would make life complicated to do it any other way. A separate one-dimensional array for each node's variables would be economical but it's hard to see how you would DIMension them all in BASIC without knowing how many nodes the program was going to have in the first place.

This way you can have up to MV variables at each node.

```
DIM V(MV,N)
```

This is the list of variable values at each of the N nodes corresponding to the variable names.

```
DIM VC(MV,N)
```

This is a, sort of, 'flag' array marking which variables have been input into the decision process and which haven't. It has values of 0 for elements which have been input and 1 for elements the system's still waiting for. The sequence is that you name the variables V$(x,y). The system asks for a value of some V$(x,y) for which VC (x,y)=1 and it puts this value in V(x,y) and, then, sets VC(x,y)=0.

DIM RV(MV,N)

This is the array which holds a 'rating' on each of the variables at each of the N nodes. The system uses this to judge which variable it wants a value for next. The V$(x,y) it asks for will, typically, be the one with the greatest corresponding RV(x,y).

DIM M(2,MV,N)

This array holds the minimum and maximum values of the variables V$(x,y). So, the minimum value that V(x,y) can have is given by M(1,x,y) and the maximum value it can have is given by M(2,x,y). You, personally, have to provide these values - usually at the time you name the variables V$(x,y). They are used to help the expert assess the relative importance of each variable in coming to decisions and are needed to calculate RV(x,y). If you don't know the minimum and maximum values the variables can take you could try guessing. After all, there must be a limit to them somewhere.

DIM VA(N)

This holds the number of variables that occur at each of the N nodes. The system already knows the maximum number that can occur (MV). But it must know how many actually do occur. This is where it's held. In a single node situation - such as we've dealt with already, this problem was solved with the single variable V. So, for several nodes, you can get rid of V and use VA(x). You can't use V(x) because you already have V(x,y) and you'd get an error if you had two arrays with the same name. If you've written a single-node expert and decide to convert it to several nodes you can save a bit of sweat by still using the variable V but, before you use it to any great extent, put V=VA(x) assuming that you're in a piece of code which is working on node x.

DIM Q(N)

This holds the number of outcomes that can occur at each of the N nodes. It's much the same as for the comments on VA(N). Similarly, you can carry on using the old variable Q for the number of outcomes but you need to put Q=Q(x) when you're working on node x.

DIM Q$(MQ,N)

This is the list of all of the outcome names that the expert system uses, just like V$(MV,N) was the list of all of the variable names.

Just as with the variable names, MQ is the maximum number of outcomes at any one node and you should:

INPUT "WHAT IS THE MAXIMUM NUMBER OF OUTCOMES AT ANY ONE NODE?"; MQ

early on in setting up the system.

Much the same comments apply here as to V$(MV,N)

DIM R(MV,MQ,N)

This is really the heart of the system, the array of rules the expert uses to make judgements. There are N sets of rules, one set for each node. Each set of rules has VA(x) variables going to it and Q(y) outcomes leading away from it. R(MV,MQ,N) is set up by training each node either from examples worked on the keyboard or by giving the system a whole list of examples and letting it work through them by itself.

DIM D(MQ)

This holds the values formed for each rule on each outcome at any one node. You could DIM D(MQ,N) for the N nodes so you always have everything to hand. It depends a bit on how you choose to search through the nodes as to whether this is necessary or not. If you get the expert to solve one node at a time then you don't really need the extra dimension and, anyway, you could always calculate D(y) afresh every time the system went on to a new node.

DIM PD(MQ)

This holds the possible values for each rule. Suppose, at a given node, only some of the variables have been entered. Then, for these variables, D(y) can be calculated from V(x,y) and R(x,y , node). For these variables D(y) and PD(y) are the same. But, for the unknown variables PD(y) can be 'guessed' by the system from the minimum and maximum values you placed in M(1,x,node) and M(2,x,node). The idea is that a 'best' guess will be made by the system on the basis of the maximum value in D(y) so far. PD(y) will then be calculated assuming that all the rest of the variables will either take their minimum or maximum values in an overall attempt to discredit the current best guess. If the system can't discredit the current choice in this

way an outcome can be chosen for the current node. If it's possible to discredit the choice by finding a different variable with a maximum in PD(y) then, obviously, more values for other variables are still needed.

DIM E(MV+1,50,N)

This holds any examples you might want to give the expert to practice on. There is room for a set of up to 50 examples on each of the N nodes.

Obviously you might want a different number of examples to 50 and you could think of a different way of holding these examples. Disc files spring to mind if you've got disc. If you're short of memory space and haven't got disc you could always DIM E(MV+1,x) where x is the number of examples you want to hold and you then keep on loading the array from cassette for each node, one at a time.

Either way, the system will want to know how many examples there are in this batch so you might as well keep this number in E(MV+1,50,N+1) and update it every time you add some more examples. If you're working a training session from the keyboard you could place every example you give the system in this array so that it can work over them later in the privacy of its own chips. This will save you wearing your fingers out entering the same examples twice.

Suppose that you've built a single-node expert and want to convert it to a multi-node expert. First, contrive an example and run it as a single-node example. This will serve as a reference for the future behaviour of the system.

Next, go through the list of variables and convert the program to N nodes. Insert extra code to ask about N, MV, and MQ. Place one big loop around the main code to step through the nodes:

FOR NO=1 TO N

main body of code

NEXT NO

Now run your single-node example again with N=1. The results should be exactly the same as before. You should have made no noticeable progress. Actually, of course, the results won't be the same for the program will go wrong, thereby announcing that you have made a mistake. So, correct the mistakes. You've probably forgotten to convert some V or Q to VA(x) or Q(y).

If, by some quirk of fate, you get the program working on one node then run it with two nodes - and put your previous example in the second node this time. As before, nothing should change. You'll be astonished and amazed at how much can change on these little occasions when nothing should have changed. It's quite an education, really.

Now run it again with two nodes. This time put your example in both nodes. What you should notice is - nothing should change. So correct the code until it doesn't.

Now comes the big moment when you start fastening these nodes together. Try:

When you've found an outcome for node NO:

```
FOR I= TO N
FOR J=1 TO VA (I)
IF Q$ (outcome,NO) = V$(J,I) THEN:V(J,I) =1:VC(J,I)=0
FOR K=1 TO Q(NO)
IF V$ (J,I)=Q$(K,NO) AND K < >outcome THEN: VC(J,I)=0
NEXT
NEXT:NEXT
```

What this does it to take the current outcome from node NO and search through the entire list of variable names to see if it matches the variables being input to any other node. If it does then it sets that variable value to 1 and sets the flag for that variable VC(J,I) to 0. The value 1 is pretty arbitrary, corresponding to Yes/No states. You could use some other value if you wanted to - for instance, the maximum value M(2,J,I).

It also searches through all of the outcomes on node NO which didn't occur. The idea being that all of the outcomes are mutually exclusive so, having got one outcome, it can't have the other outcomes and VC(J,I) can then be set to zero so that the expert doesn't go asking for values for them.

Also, when you privide a variable value, V(VV,NO) you could

```
FOR I= TO N
FOR J=1 TO VA(I)
IF V$(VV,NO)=V$(J,I) THEN: V(J,I) =V(VV,NO):VC(J,I)=VC(VV,NO)

NEXT:NEXT
```

So that if any given variable occurs as an input to more than one node its value won't be requested more than once.

The whole idea is to squeeze as much out of the information you're giving as possible. If you provide a variable value it has to work for its living by providing inputs to as many nodes as it can. If an outcome is chosen this should be fed into as many nodes as need that information as quickly as possible in the hope of making further outcomes possible with as little intervention from you as possible.

If you now set up the system again with two nodes and make each of the two nodes identical, containing your previous example, you should find that, having solved the first node, the expert immediately gives the answer to the second node. The same answer with no more help from you - after all, it's got all of the information for the second node when it solved the first node.
This should leave you with a basic multi-node system which is adequate for a number of problems.

Try setting it up with an example. Say, two nodes:

Node 1

Variables	*Outcomes*
Feathers	Animal
Metal	Machine
Beak	

Node 2

Variables	*Outcomes*
Animal	Bird
Machine	Plane
Beak	Glider
Engine	

The exact details of the example don't matter too much. The main points to note are that it should first decide whether or not it's being given an animal or a machine. It should then use this information to decide whether or not it's being given a bird, plane or glider. It should not have to ask about Beak twice (you provide that information in node 1). And it should not have to ask about Animal or Machine at all (it should be able to work that out for itself in node 1).

7.3 Going through the nodes

Suppose that you decide to set up an expert system with more than one node in it. The expert would then have several points at which it became able to pass some kind of judgement and, depending on each judgement, it might then be able to go on to make more judgements.

The problem that you face when you're building a system like this is that the program has to pass from one node to another. You have to give it a method for deciding which node it should go to next, and the method you choose will influence how well, or badly, the system works.

To make matters worse, there isn't really any ideal method for every situation. Depending on what you want to do with the expert system one method might be better than another.

It's exactly the same as the problem of tree-searching. Or graph searching. And, to really go into it in detail needs the best part of a book on the subject.

Fortunately, the way we've worked so far has left us with a pretty loose sort of problem so we might as well attack it in a pretty loose sort of way.

For a start, you could visit the various nodes in exactly the same way as you selected variables in the single node case.

Recall, that we had an array RV(V) which held a value for each of the V variables. This value was calculated as being the maximum amount of variation that each variable could introduce into the problem. Take the minimum and maximum values for a variable, calculate the difference between them, and then go to the rule array and look at the values in the rule array for that variable. Multiply them together for each outcome and that gives some sort of measure of the likely importance of each variable. Take the most important variable and let the expert ask for a value for that variable first. It that doesn't enable it to make a judgement, then go and get the next most important variable. And so on.

There's no reason why we shouldn't use the same approach over a large number of nodes. There would be some difference in the programming because some variables might occur in more than one node - but, generally, that would make them seem rather more important to the expert and it would ask about these first. Which seems a reasonable approach. It simply consists of selecting for study those variables which provide the most information to as many nodes as possible.

Actually, what we're doing if we do this is to turn ordinary tree searching

techniques on their head a bit. Normally, one would specify a method of searching a tree with reference to the structure of that tree. You know the sort of thing - search down one path until you get to a dead end, then search down the next path to that, and so on.

In this case we're not searching in that way at all. We have labels on each path corresponding to the names of the variables we've provided. What we're saying is: find the most important variable and then explore all the paths that use that variable. And there's something to be said for doing this. For, in general, we don't know the exact form of the problem you might set your expert so we don't know the best searching method through that problem. We do, however, know that it must have variables in the problem so we ignore the structure and concentrate on the variables themselves. This approach is often called 'Forward Chaining'.

It's a method that will eventually produce answers, even if it means a value has to be provided for each and every variable before you get the specific answer you want. It's just that some people might feel uneasy about a method that seems to proceed with complete disregard for the network through which it's proceeding or the eventual goals you might wish to achieve. Not so much a method, it's rather more like some kind of primordial sludge which oozes through the network of nodes on the basis of: "This bit looks interesting, let's look at that next."

But, as the system ignores the particular way the nodes are organised it also leaves you free to ignore the way they're organised too. All you've got to do is to specify your nodes and name variables and outcomes for each one, and the system will get somewhere in the end. It does have a number of practical disadvantages though.

One of the most practical disadvantages is the fact that all linkage between nodes is done by giving variables and outcomes specific names. This means that if you happen to forget exactly how you spelt some variable last time the required linkage won't be made. This is certainly a bit like saying that if you press the wrong keys on your computer you'll get the wrong answers. But if you have a large number of variables and you've had a tiring day it can be a real problem. There isn't much that can be done about it in principle. In practice though, it's a good idea to get the system to let you name variables by showing you a list (menu) and asking which variable you want, rather than by typing it in (sometimes incorrectly) each time.

More specific to this method of working is the fact that the expert might consider a certain variable to be important and, so, ask you for a value for that variable. You then realise that you don't really want to be asked about that variable at all. This might be because you don't have a value for it at the moment or because you know, even if the expert doesn't, that it just isn't

useful to provide that information for this particular problem. You'd rather it had asked about something else.

This problem can be fairly easily overcome with several nodes. Suppose you have N nodes you can access them by placing them inside a loop. So the system considers node 1 first, node 2 next and, so, node N last.

At each node, the system could use the previous method for deciding which variables to ask about and, on finding an outcome, it could check all other nodes to see if that outcome was an input variable anywhere else. But, the exact order of passing from one node to the next could be rigidly specified at the time you set the system up. This way, if you know the problem you want to train the expert on, you could set it up to deal first with those things that you, personally, know should be handled first. And if you don't know the perfect order, or don't care, then nothing's really lost by it.

Roughly speaking, this problem is the problem of where to start the expert working on the task in hand and it's at its worst if you don't know where to start the task either. Not a trivial point. Consider: the system could look for the most important variables and ask for values on those, but what seems important to it might not seem important to you. Or you could specify some starting point by means of an orderly progression through the various nodes. But, suppose you sometimes have one lot of variables and you sometimes have another lot. In general, you don't have values for all the variables and, depending on what you do have, the expert should obviously start from different points and then pass to different points.

Ideally you just want to be able to go to the expert and give it what information you have and let it make what it can of it. This you can do by giving the expert a variable name and a value for that variable. The variable name then determines which node or nodes the system goes to first. And, because of the danger of mis-keying the variable name you should either select them from the menu or check that the input corresponds to an existing variable before it gets accepted into the system and clogs everything up.

If that was the problem of where the expert should start working, there's an equal problem of where the expert should finish up in the end.

It's fine to say that eventually it will have worked out everything it can so you'll always get an answer to your problem. In practice you don't always want to sit around entering data for various nodes which are really quite incidental to the main problem.

The method of working through the nodes in order allows you to control this to some extent inasmuch as node N could contain the final, overall,

decisions and the aim is then to get to node N as quickly as possible. In general, what you can do is to always go to the node which is nearest to node N. And the problem is then to define 'nearest'.

The simplest way is to think of each node as either 'active' or 'dead'. If it's already produced an outcome it's dead - you don't need to go through it again. If it hasn't it's active - and you have to find an outcome for that node. You can then travel in order through the nodes, resolving the active ones to take you successively nearer to node N. At which point it's easy to observe that node N is the nearest node to node N - so why not try to resolve that one first? It would save so much time.

Well, if you could do that, you wouldn't have needed the other nodes in the first place. And, if you can't do that, where can you start in order to finish there? It's at about this point that you realise that searching through nodes isn't necessarily an easy subject to cover completely.

The most general method takes us back to you supplying details of what variables you have values for, entering those, and seeing what outcome turns up.

But, if you don't want to enter more details than you have to and you have specific ideas about which intermediate results you'd like the picture gets more complicated. Suppose, for instance, that you simply want the quickest route from some starting point to a final solution. Does that mean, by quickest, that the smallest number of nodes should be visited? Or the least number of questions asked? It's up to you what the answer is to that question - but it does show that there isn't just one, single, answer. Asking the smallest number of questions gives you the least amount of work to do. Visiting the smallest number of nodes gives you the smallest number of intermediate results. Maybe you'd like to get as many intermediate results as possible for the least effort. So you'd like to visit as many nodes as possible whilst the system asks you as few questions as possible. And, possibly, these aims are incompatible.

At the risk of increasing the programming effort you could keep track of the distance to node N, in terms of nodes and in terms of variables still unresolved, for each point in the network of nodes. But the increase in programming effort may be daunting. This is especially the case if there's more than one route to node N from some points in the program. You not only have to find out what the 'distance' is for one route you have to find out how many routes there are and how far they are from node N and, somehow, you have to keep track of all these routes so that you can choose one which suits you best.

This isn't an impossible problem by any means. If you go through the

literature on graphs and trees you'll find plenty of solutions. But it certainly isn't trivial.

In general, the programming effort will be much reduced, and the strain on the intellect lessened, if you proceed by one of the following methods:

Volunteer the information to the system. Specify on which variables you have information and let the expert proceed on all nodes that are applicable to that information to see what conclusions it can reach.

Let the system ask for information on those variables which seem most important to it. Assuming that you have all the information it's likely to want you can provide what's requested and let the system draw what conclusions it can until it either gets the answer you want or information has been provided on all variables.

Get the system to pass from node to node sequentially. This way it won't miss anything and will get to the end eventually. When you first set the system up you should try to establish the nodes in an order that seems sensible to you. You might as well let the system make what deductions it can, when it can, with this method though - because it might happen to get to the end by some route faster than you expected. And, if the system comes to a node which has, by this means, already produced an outcome it might as well skip it otherwise it's doing the same job twice.

It's worth mentioning that you could do exactly the same with a single-node system. Or a system which looks like a single node. If, say, you just have R(I,J) to represent one node, you could have a variety of possible outcomes some of which could be identical to some of the variables.

If the expert decided on some outcome and this happened to match up with an input variable this outcome could be used to switch on, as it were, that input variable. After which a flag would have to be set on that outcome so that its effects weren't counted in more than once.

You can then proceed by specifying variable names on which you intend to provide information, or letting the system pick its own variables on which it wants information, or proceeding sequentially through all variables in turn.

The problem with this approach - of having all the separate nodes effectively contained in one big node - is the scope it offers for getting confused. Suppose, for instance, the system was meant to predict the weather and, for a start, it judged that it was a warm day today. What's to stop it then making the comment that it was *not* a warm day today? After all, with a given set of rules the *two* most likely outcomes might be

mutually exclusive and it might give both these outcomes before it gives anything else.

Of course, you can easily devise a method for dealing with this problem. But avoiding confusion is one of the best ways of getting programs to work and a less than ideal method in some respects quickly becomes the best method in every respect if it happens to be the only one you can get working reliably.

7.4 Tailor-made nodes

On the day when you're setting up your expert system for a particular problem you might be faced with the odd node which seems to be either too complicated or too simple for you to want to just let it learn by example.

For instance, suppose that you've decided to set up an amazing medical diagnosis system. One of the input variables is the patient's temperature. The first thing you want to do is to get the system to say whether or not the patient actually does have a high temperature or not. Now - that's easy. If it's over, say, 99 degrees Fahrenheit we want outcome 1 at some node - outcome 1 being High Temperature. Clearly, if you give the expert enough examples to train on it will eventually learn to give this information of its own accord. But it might seem like rather a lot of hard work. After all, you actually, in this case, know what the rules should be. Why mess around with a training session?

Quite. So, skip the training session and set up a tailor-made node to assess the patient's temperature. The problem is simply to find values for R(I,J) which will give the right answers. Here's a node that will do it:

	High Temperature	Low Temperature
Constant	0	99
Temperature	1	0

This uses a node of size R(2,2). Suppose now that you have an input V(I). V(0) contains the value 1, the constant, and V(1) contains the patient's temperature. Multiplying by the first rule, high temperature, the answer is always equal to V(1). Multiplying by the second rule, low temperature, the answer is always equal to 99. Obviously, the first rule gives the highest value as soon as V(1), the patient's temperature, is over 99 - and, so, the system will always get it right about the matter of high versus low temperature. Much easier than giving the thing a whole load of examples and waiting for it to sort out the answer on its own.

In general, the decision rule always gives an answer which depends on:

$$D = b_1x_1 + ... + b_nx_n$$

so, if you can put the decisions into this form it's fairly straightforward.

If you can't think of a way of expressing the problem in this style then you can always give the system some examples and see if it can work something out for you. If what you want the system to do is particularly complicated then you've got problems - a comment which isn't particularly encouraging because there isn't, in general, any method for fitting everything you might want to do into this form.

Suppose you had an input variable which could take values from 0 to 100. If it falls in the range 50 to 60 then this has some special significance for you and you want the expert system to detect this significance. It's not an unusual problem - it might be some measurements you're taking and 50 to 60 might be optimal values. Now, it isn't immediately apparent that a set of rules could be written to detect this particular band of values. You could get around the problem in two ways. First you could use two nodes to deal with it. The first node could detect if the value was over 50 - in just the same way as we set up a node to detect a high temperature. If it is over 50 the system could pass its result to another node which, in the same way, would detect if the value was under 60. If the system gets the right outcome from both of these nodes it can then deduce that the variable is in the range 50 to 60 and it's solved your problem. So it is possible to detect the required values using this framework.

On the other hand, it would seem to be so much simpler to put in a piece of code:

```
IF V(I) >= 50 AND V(I) <= 60 THEN: Q(J)=1:
```

and so on, and so forth rather than to force the problem into a rather alien framework.

But once you've started to do this sort of thing you've still got the problem remaining of how to fit odd items like this into the overall pattern. The simplest way is to place tailor-made pieces of code like this very early on in the program so that every time the program starts to work on a new node, say, the variable values for the tailor-made items are checked to see if they've been provided yet. This means that the code is likely to get executed more often than is strictly necessary - but it has the important advantage of ensuring that the expert doesn't forget to execute them at all!

Suppose now, thinking back to our amazing medical diagnosis problem, that we want to decide if a patient has a fever. We define a fever as a high temperature and flushed cheeks (or any other set of symptoms that you'd

like to choose). The system has already decided about the patient's temperature so all that's left is to see if he has flushed cheeks as well. Given enough examples, the expert will work out a set of rules for this by itself. But equally, you can put in your own rules. And this is an example of logical connectives: if (high temperature) and (flushed cheeks) ...

Suppose that we had rather different medical views we might reckon that it was sufficient for the patient to have just one of those symptoms: if (high temperature) or (flushed cheeks) ...

The logical connectives 'and' and 'or' are so common that it's worth having some tailor-made rules for them.

	Outcome 1	Outcome 2
Constant	0	n-1/2
Variable 1	1	0
Variable 2	1	0
.		
.		
.		
Variable n	1	0

Using this set of rules outcome 1 gives a maximum value of n, if all n variables have the value 1. Outcome 2 gives the same value irrespective of the variable values it gets and the value it gives is always n-1/2. So, outcome 2 will always be chosen by the expert except in the special case when all n variables have the value 1. In other words, when we have (variable 1) and (variable 2) and (variable 3) ... and ... (variable n) then outcome 1 will be chosen.

	Outcome 1	Outcome 2
Constant	0	1/2
Variable 1	1	0
Variable 2	1	0
.		
.		
.		
Variable n	1	0

Using this set of rules outcome 2 always gives the value 1/2. Outcome 1 will be less than this, with a value 0, as long as all of the variables have the value 0. As soon as any one variable produces a value of 1 outcome 1 gives the highest value. So, outcome 2 will always be chosen by the expert until we have:

(variable 1) or (variable 2) or ... or (variable n).

You can see that these two sets of rules are very similar to each other. In fact, if the constant on outcome 2 has the value (x-1/2) the expert will reckon on outcome 1 as soon as x or more of the input variables have the value 1 instead of 0. For the (and) connective x=n while for the (or) connective x=1. But you could have any other number if it suited your application.

The advantages of setting up your own rules like this are that you know exactly what the system's going to do at any moment (or, roughly, anyway). The advantages of letting the system learn its own rules by example are that it provides a relatively easy way of entering the rules into the system coupled with the bonus that you don't have to bother to sit down and work out in advance what those rules should be.

7.5 Specific code

Thinking about amazing systems for medical diagnoses, one ponders on the matter raised in the last section. The matter of fever. Suppose we reckoned a fever as a high temperature and flushed cheeks. Well, we saw how we could, either explicitly or by example, build this into our knowledge base in the form of rules and how a fever could then be diagnosed. But wouldn't it have been simpler just to write the BASIC line:

```
IF T > 99 AND F=1 THEN: PRINT "FEVER"
```

where T contains the patient's temperature and F=1 if the patient has flushed cheeks?

Well, yes, actually, it would be much easier. You could scarcely go wrong at all.

In fact, you could have a whole collection of statements like this and they'd give you a system to diagnose anything (as long as the 'anything' was covered by your programming). It would be quicker and easier to do things this way because BASIC is such a straightforward programming language that, if you've had a bit of practice, you can just sit down and write the code almost as quickly as it would take to set up a set of rules in a more general purpose system. It might be a bit haphazard until you'd defined the program but a general purpose approach has to be a bit haphazard too just to make sure it covers all possible uses it might be put to.

It's all to do with a theoretical claim that's sometimes made about expert systems. It's said that, ideally, they're general purpose programs. They contain a method for making expert judgements in general. And it's

suggested that, with an expert system, you add a knowledge base and get a different kind of expert. Now that is something you can't do if you just sit down and write your own code for a specific problem. If you write a medical diagnosis system just by sitting down and writing it you'll never be able to alter it readily to solve another problem.

All of which is hardly a fair criticism if that's what you want to do. After all, most expert systems only really work on one specific problem - they aren't truly general purpose. The one you're building might be general purpose, able to do anything. But there aren't many others like it. And, as we've seen so far, it does have its limitations at times.

7.6 Saving your expert

Come the end of the day, you've been sitting at the keyboard training your expert to a high degree of skill and it's time to go to bed. Switch the machine off and - lo and behold! - the expertise vanishes like mist.

You knew this would happen, after all it was only data. But it's no use having an expert that has to be rebuilt from scratch every time you want to run it. You want it to get better, not worse, every day.

The program can be SAVE'd on cassette or disc but exactly how you save the data depends on your system. With a disc-based system you might have found it better to use file-handling techniques rather than all those arrays (now he tells us!) but you couldn't have done that if you didn't have discs and you might not have discs. Besides, it depends largely on your system how many files of what type could be open at any one time.

The simplest way is to dump the whole memory of the machine either to cassette or disc, depending on what you've got. Again, it depends on your system as to whether this will work. Before you start building up the expert too thoroughly try dumping the memory. Do this in the middle of a program run using whatever method your manual suggests might work. If all is well you should be able to switch the machine off, then on again, reload the memory and carry on from where you were. If you can, it's fine to use this method. If you can't do exactly that, then see if you can get the program running again anyhow - maybe from some other point than the one you broke off from. Try a GOTO statement to get into the program rather than a RUN statement.

If this doesn't work you've got to think of something else. That something else is likely to consist of picking on specific arrays and explicitly SAVE'ing them. For instance, you'll want R(I,J) again for the rules and M(2,J) for the minimum and maximum values of the variables.

What you can do is to write a small routine which saves the program and all of the variable arrays you want. At the end of a day's session you go to this routine which will be, for the Apple II:

SAVE:STORE R:STORE M and so on

and execute it. Either to cassette or disc this should lay down the program and variables just in the order shown.

Then, when you start up the next day, you LOAD the program and key RUN immediately. Have as the first instructions in the program RECALL statements in the same order as the STORE statements you had last. This will bring the array information in off the cassette or disc in the order you stored them.

On the Sinclair Spectrum, you SAVE the entire program complete with all data including arrays, with a statement such as

SAVE "EXPERT"

But, when you re-load the program, you must not type RUN because that clears all variables. Instead, type GOTO 10, say where 10 is a line number. This technique is used in the listing in section 7.7

7.7 The multi-node code

To save anyone the inconvenience of having to do any thinking, here's a complete, menu-driven, multi-node, learning expert system which should enable anyone, after a few seconds practice, to achieve some reasonably modest aims. Like, for instance, World Domination (as long as you can think of some examples to give it).

Fig. 7.1
Menu-driven, multi-node expert

(a) Apple II listing

```
10  Z = 0:W = 1:TW = 2:MI =  - 1E38:TH = 13:TE = 10:FO = 40
20  GOSUB 1200
30  PRINT : PRINT : PRINT "1. INITIALISE EXPERT"
40  PRINT "2. INPUT EXAMPLES"
50  PRINT "3. EXERCISE EXPERT"
60  PRINT "4. TRAINING SESSION"
70  PRINT "5. NORMAL RUNNING"
80  PRINT "6. SAVE CURRENT EXPERT": PRINT "7. EXAMINE RULE VALUES AND
    EXAMPLES"
90  PRINT : PRINT : INPUT "CHOOSE AN OPTION";OP
100 IF OP = W THEN 170
110 IF OP = TW THEN 680
120 IF OP = 3 THEN 970
130 IF OP = 4 THEN :T$ = "Y": GOTO 330
140 IF OP = 5 THEN :T$ = "N": GOTO 320
150 IF OP = 6 THEN 1100
160 IF OP = 7 THEN : GOSUB 1120: GOTO 20
170 GOSUB 1200: PRINT "TO SET UP AN EXPERT SYSTEM": PRINT "PLEASE ANSWER
    THE FOLLOWING :-": PRINT : PRINT : INPUT "HOW MANY NODES HAVE YOU ?";N

180 PRINT : PRINT : PRINT "WHAT IS THE MAXIMUM": PRINT "NUMBER OF VARIABLES":
    PRINT "AT ANY ONE NODE ?": INPUT MV
190 PRINT : PRINT : PRINT "WHAT IS THE MAXIMUM": PRINT "NUMBER OF OUTCOMES":
    PRINT "AT ANY ONE NODE ?": INPUT MQ
200 DIM V$(MV,N),V(MV,N),VC(MV,N),RV(MV,N),M(TW,MV,N),E(MV + W,50,N),VA(N),
    QU(N), Q$(MQ,N),R(MV,MQ,N),D(MQ),PD(MQ),QC(MQ,N),EZ(N)

210 FOR NO = W TO N: GOSUB 1200: PRINT "NODE ";NO: PRINT : PRINT "HOW MANY
    VARIABLES HAVE YOU": INPUT "AT THIS NODE ?";VA(NO)
220 PRINT : PRINT "PLEASE NAME THESE VARIABLES :-": PRINT
230 FOR I = W TO VA(NO): VTAB (TH): FOR P = W TO TE: PRINT SPC( FO);: NEXT : VTAB
    (TH): PRINT : PRINT "VARIABLE ";I;" IS ";: INPUT V$(I,NO):X$ = "N"
240 FOR P = W TO NO: FOR Q = W TO VA(P): IF V$(I,NO) = V$(Q,P) AND NO < > P
    THEN : PRINT "(";V$(I,NO);") ALREADY OCCURS": PRINT "ON NODE ";P: PRINT
    "WITH MINIMUM VALUE ";M(W,Q,P): PRINT "AND MAXIMUM VALUE ";M(TW,Q,P):
    PRINT : INPUT "IS IT THE SAME HERE ?";X$
```



```
250 IF X$ = "Y" THEN :M(W,I,NO) = M(W,Q,P):M(TW,I,NO) = M(TW,Q,P):X$ = "N":
    GOTO 280
260 NEXT : NEXT
270 PRINT : PRINT : PRINT "IT HAS MINIMUM VALUE = ";: INPUT M(W,I,NO): PRINT "IT
    HAS MAXIMUM VALUE = ";: INPUT M(TW,I,NO):
280 NEXT I
290 GOSUB 1200: PRINT "NODE ";NO: PRINT "HOW MANY OUTCOMES HAVE YOU ?":
    INPUT "AT THIS NODE ?";QU(NO)
300 PRINT : PRINT "PLEASE NAME THESE OUTCOMES :-": PRINT : FOR I = W TO QU(NO):
    PRINT "OUTCOME ";I;" IS ";: INPUT Q$(I,NO): NEXT
310 NEXT NO: GOTO 20
320 IF T$ = "N" THEN : GOSUB 1200: PRINT "EXPERT READY": PRINT : PRINT : PRINT "I
    WILL ASK YOU FOR INFORMATION": PRINT "AND TRY TO MAKE A CORRECT
    DEDUCTION": GOTO 350
330 GOSUB 1200: PRINT "TRAINING SESSION": PRINT : PRINT : PRINT "I WILL ASK YOU
    TO ENTER SOME VARIABLES": PRINT "THEN I WILL MAKE A DEDUCTION": PRINT
    "YOU MUST TELL ME IF I AM RIGHT"
340 PRINT : PRINT "PRESS ANY KEY TO CONTINUE": GET X$
350 FOR NO = W TO N: FOR I = W TO VA(NO):VC(I,NO) = W:V(I,NO) = Z: NEXT I: FOR J
    = W TO QU(NO):QC(J,NO) = W: NEXT J: NEXT NO: FOR NO = W TO N:V =
    VA(NO):Q = QU(NO): FOR J = W TO Q:D(J) = Z
360 FOR K = W TO V: IF VC(K,NO) = Z THEN :D(J) = D(J) + V(K,NO) * R(K,J,NO)
370 NEXT K:NEXT J:D=MI:FOR J=W TO Q:PD(J)=D(J):IF D(J)>=D THEN:HJ=J:D=D(J)
375 NEXT J
380 GOSUB 1200: PRINT "NODE ";NO
390 H = Z:VV = Z: FOR I = W TO V:RV(I,NO) = Z:M = Z:N1 = Z: FOR J = W TO Q:M = M +
    R(I,J,NO) * VC(I,NO) * QC(J,NO):N1 = N1 + QC(J,NO): NEXT J: IF N1 THEN :M = M /
    N1
400 FOR J = W TO Q:RV(I,NO) = RV(I,NO) + ABS (M(W,I,NO) - M(TW,I,NO)) * ((R(I,J,NO)
    - M) ^ TW) * VC(I,NO) * QC(J,NO): NEXT J: IF RV(I,NO) > H THEN :VV = I:H =
    RV(I,NO)
410 NEXT I: IF H = Z THEN 560
420 VC(VV,NO) = Z
430 FOR J = W TO Q - W: FOR K = J + W TO Q: IF R(VV,J,NO) < > R(VV,K,NO) AND
    QC(J,NO) AND QC(K,NO) THEN 450
440 NEXT K: NEXT J: GOTO 390
450 PRINT VV;". ";V$(VV,NO);: INPUT V(VV,NO): IF V(VV,NO) <M(W,VV,NO) OR
    V(VV,NO) > M(TW,VV,NO) THEN : PRINT : PRINT "INPUT IS OUT OF RANGE": PRINT
    "YOU GAVE MINIMUM ";M(W,VV,NO): PRINT "AND MAXIMUM ";M(TW,VV,NO):
    GOTO 450
460 FOR P = W TO N: FOR V1 = W TO VA(P): IF V$(V1,P) = V$(VV,NO) THEN :V(V1,P) =
    V(VV,NO):VC(V1,P) = Z
470 NEXT V1: NEXT P:D = MI: FOR J = W TO Q:D(J) = D(J) + V(VV,NO) * R(VV,J,NO):PD(J)
    = D(J): IF D(J) > = D THEN :D = D(J):HJ = J
480 NEXT J
490 FOR I = W TO V: FOR J = W TO Q
500 IF VC(I,NO) = W THEN : IF R(I,J,NO) > R(I,HJ,NO) THEN :PD(J) = PD(J) + (R(I,J,NO) -
    R(I,HJ,NO)) * M(TW,I,NO)
510 IF VC(I,NO) = W THEN : IF R(I,J,NO) < R(I,HJ,NO) THEN :PD(J) = PD(J) +
    (R(I,J,NO) -  R(I,HJ,NO)) * M(W,I,NO)
520  NEXT J: NEXT I
530  H2 = MI: FOR J = W TO Q: IF PD(J) > = H2 THEN :H2 = PD(J):HI = J
540  IF PD(J) < PD(HJ) AND J < > HJ THEN :QC(J,NO) = Z
```

```
550  NEXT J: IF PD(HI) < > PD(HJ) THEN 390
560  IF HJ AND T$ = "Y" THEN : PRINT : PRINT "CAN I DEDUCE OUTCOME :";Q$(HJ,NO):
     INPUT A$: IF A$ = "Y" THEN : GOSUB 780: GOSUB 850: NEXT NO: PRINT : INPUT
     "DO YOU WISH TO CONTINUE TRAINING ?";B$: IF B$ = "Y" THEN 350
570  IF T$ = "N" THEN : PRINT "I SUGGEST ";Q$(HJ,NO);" AS LIKELY": GOSUB 780: NEXT
     NO: PRINT : INPUT "DO YOU WISH TO CONTINUE ?";B$: IF B$ = "Y" THEN 350
580  IF (A$ = "Y" AND B$ = "N" AND T$ = "Y") OR (B$ = "N" AND T$ = "N") THEN 20
590  IF HJ = Z THEN 600
600  FOR I = W TO Q: PRINT I;". ";Q$(I,NO): NEXT I: PRINT : INPUT "WHICH OUTCOME IS
     IT ?";HI
610  FOR I = W TO V: IF VC(I,NO) THEN : PRINT "WHAT VALUE WAS ";V$(I,NO);" ?";:
     INPUT V(I,NO):VC(I,NO) = Z
620  NEXT I
630  FOR I = W TO Q: IF D(I) > = D(HI) AND I < > HI THEN: FOR J = W TO V:R(J,I,NO) =
     R(J,I,NO) - V(J,NO): NEXT J
640  NEXT I
650  FOR J = W TO V:R(J,HI,NO) = R(J,HI,NO) + V(J,NO): NEXT J:HJ = HI: GOSUB 780:
     GOSUB 850: NEXT NO
660  PRINT : PRINT : INPUT "DO YOU WISH TO CONTINUE TRAINING ?";C$: IF C$ = "Y"
     THEN 330
670  GOTO 20
680  FOR NO = W TO N: GOSUB 1200: PRINT "NODE ";NO: PRINT : PRINT : PRINT "INPUT
     SOME EXAMPLES": PRINT : PRINT : PRINT "HOW MANY EXAMPLES HAVE YOU ?":
     INPUT "AT THIS NODE";N2
690  FOR I = W TO N2
700  GOSUB 1200: PRINT "EXAMPLE NO. ";I;" ON NODE ";NO: PRINT : PRINT
710  FOR J = W TO VA(NO)
720  PRINT "VARIABLE ";J;" (";V$(J,NO);") IS ";: INPUT E(J,I + EZ(NO),NO)
730  NEXT
740  PRINT : PRINT "AND THE OUTCOME IS ?": FOR J = W TO QU(NO): PRINT J;".
     ";Q$(J,NO): NEXT : INPUT "ANSWER BY NUMBER :";E(VA(NO) + W,I + EZ(NO),NO)
750  NEXT
760  EZ(NO) = EZ(NO) + N2
770  NEXT NO: GOTO 20
780  FOR P = W TO N
790  FOR V1 = W TO VA(P)
800  IF V$(V1,P) = Q$(HJ,NO) THEN :V(V1,P) = M(TW,V1,P):VC(V1,P) = Z
810  FOR R = W TO Q: IF V$(V1,P) = Q$(R,NO) AND R < > HJ THEN :VC(V1,P) = Z
820  NEXT
830  NEXT : NEXT
840  RETURN
850  INPUT "DO YOU WISH TO KEEP THIS EXAMPLE ?";E$: IF E$ = "N" THEN : RETURN
860  FOR J = W TO VA(NO): IF VC(J,NO) = Z THEN 900
870  VV = J:VC(J,NO) = Z
880  PRINT "WHAT VALUE WAS :-"
890  PRINT VV;". ";V$(VV,NO);: INPUT V(VV,NO): FOR P = W TO N: FOR Q = W TO VA(P):
     IF V$(Q,P) = V$(VV,NO) THEN :V(Q,P) = V(VV,NO):VC(Q,P) = Z
895  NEXT Q:NEXT P
900  NEXT J
910  EZ(NO) = EZ(NO) + W
920  FOR J = W TO VA(NO)
930  E(J,EZ(NO),NO) = V(J,NO)
```



```
940  NEXT
950  E(VA(NO) + W,EZ(NO),NO) = HJ
960  RETURN
970  GOSUB 1200: PRINT "EXPERT IS WORKING ON IT": FOR NO = W TO N:DE =
     EZ(NO):I = Z: FOR P = W TO DE * 200:I = INT ( RND (W) * DE + W)
980  D = Z:C = Z:V = VA(NO):Q = QU(NO)
990  FOR J = W TO V
1000 D = D + E(J,I,NO) * R(J,E(V + W,I,NO),NO)
1010 NEXT
1020 FOR K = W TO Q
1030 D2 = Z
1040 IF K < > E(V + W,I,NO) THEN : FOR J = W TO V:D2 = D2 + E(J,I,NO) * R(J,K,NO):
     NEXT
1050 IF D2 > = D AND K < > E(V + W,I,NO) THEN :C = C + W: FOR J = W TO V:R(J,K,NO) =
     R(J,K,NO) - E(J,I,NO): NEXT
1060 IF C = W THEN : FOR J = W TO V:R(J,E(V + W,I,NO),NO) = R(J,E(V + W,I,NO),NO) +
     E(J,I,NO): NEXT :C = C + W
1070 NEXT
1080 NEXT
1090 NEXT NO: GOTO 20
1100 GOSUB 1200: PRINT "SAVE THE CURRENT EXPERT": PRINT : PRINT "CALL-151":
     PRINT : PRINT "*0000.C000W": PRINT : PRINT "TO CASSETTE": PRINT : PRINT "TO
     GET IT BACK": PRINT"*0000.C000R
1110 PRINT : PRINT : PRINT "KEY 'CONT' TO CONTINUE": STOP : GOTO 20
1120 FOR PP = 1 TO N
1130 PRINT "NODE",PP
1140 PRINT "E(I,J,NODE)"
1150 FOR QQ = W TO VA(PP) + W: FOR RR = W TO EZ(PP): PRINT E(QQ,RR,PP);" ";: NEXT :
     PRINT : NEXT
1160 PRINT "R(I,J,NODE)"
1170 FOR QQ = W TO VA(PP): FOR RR = W TO QU(PP): PRINT R(QQ,RR,PP);" ";: NEXT :
     PRINT : NEXT
1180 GET A$
1190 NEXT PP
1200 HOME : PRINT ,"E X P E R T": PRINT ,"----------": PRINT : PRINT : PRINT : RETURN : REM
     :SCREEN HEADER
```

(b) Sinclair Spectrum listing

```
    2 REM Don't forget to type GO
TO 10 to start the system withou
t losing the variables!
   10 LET mi=-1e38: LET hi=0: LET
 a$="": LET b$="": LET t$="": LE
T d=0
   20 GO SUB 1200
   40 PRINT '''"1. Initialise Exp
ert"'"2. Input Examples"'"3. Exe
rcise Expert"'"4. Training Sessi
on"'"5. Normal Running"'"6. Save
 Current Expert"'"7. Display Rul
es and Examples"'"8. Stop"
  100 INPUT "Choose an Option: ";
op
  110 IF op=1 THEN GO TO 170
  120 IF op=2 THEN GO TO 670
  130 IF op=3 THEN GO TO 970
  140 IF op=4 THEN LET t$="y": GO
 TO 330
  150 IF op=5 THEN LET t$="n": GO
 TO 320
  160 IF op=6 THEN GO TO 1100
  165 IF op=7 THEN GO TO 1120
  166 IF op=8 THEN STOP
  168 GO TO 20
  170 GO SUB 1200: PRINT "To set
up an Expert System"'"Please ans
wer the following:": INPUT "How
 many nodes have you";n
  180 PRINT '''"What is the maximu
m number"'"of variables at any o
ne node?": INPUT mv
  190 PRINT '''"What is the maximu
m number"'"of outcomes at any on
e node?": INPUT mq
  200 DIM v$(mv,n,20): DIM v(mv,n
): DIM c(mv,n): DIM r(mv,n): DIM
 m(2,mv,n): DIM e(mv+1,50,n): DI
M a(n): DIM u(n): DIM q$(mq,n,20
): DIM s(mv,mq,n): DIM d(mq): DI
M p(mq): DIM q(mq,n): DIM z(n)
  210 FOR o=1 TO n: GO SUB 1200:
PRINT AT 20,0;"At Node ";(o): IN
PUT "How many variables are ther
e?";a(o)
  220 PRINT '"Please name these v
ariables"
  230 FOR i=1 TO a(o): INPUT "Var
iable ";(i);" is ";v$(i,o): LET
x$="n"
  240 FOR p=1 TO o: FOR q=1 TO a(
p): IF v$(i,o)=v$(q,p) AND o<>p
THEN PRINT (v$(i,o));" already
occurs on node ";p'"with minimum
 value "(m(1,q,p))'"and maximum
value"'(m(2,q,p)): INPUT "Is it
the same here?";x$
```



```
  250 IF x$="y" THEN LET m(1,i,o)
=m(1,q,p): LET m(2,i,o)=m(2,q,p)
: LET x$="n": GO TO 280
  260 NEXT q: NEXT p
  270 INPUT '"It has minimum valu
e=";m(1,i,o): INPUT "And maximum
 value=";m(2,i,o)
  280 NEXT i
  290 GO SUB 1200: INPUT "How man
y outcomes "'"at Node ";(o);"?";
u(o)
  300 PRINT '"Please name these o
utcomes:": FOR i=1 TO u(o): INPU
T "Outcome ";(i);" is ";q$(i,o):
 NEXT i
  310 NEXT o: GO TO 20
  320 IF t$="n" THEN GO SUB 1200:
 PRINT "Expert Ready"''"I will a
sk you for information"'"and try
 to make a correct"'"deduction":
 GO TO 350
  330 GO SUB 1200: PRINT "Trainin
g Session"''"I will ask you to e
nter some"'"variables, then I wi
ll make"'"a deduction. You must
tell me"'"if I am right."
  340 PRINT '"Press any key to co
ntinue"
  345 LET a$=INKEY$: IF a$="" THE
N GO TO 345
  350 FOR o=1 TO n: FOR i=1 TO a(
o): LET c(i,o)=1: LET v(i,o)=0:
NEXT i: FOR j=1 TO u(o): LET q(j
,o)=1: NEXT j: NEXT o: FOR o=1 T
O n: LET v=a(o): LET q=u(o): FOR
 j=1 TO q: LET d(j)=0
  360 FOR k=1 TO v: IF c(k,o)=0 T
HEN LET d(j)=d(j)+v(k,o)*s(k,j,o
)
  370 NEXT k: NEXT j: FOR j=1 TO
q: LET p(j)=d(j): NEXT j
  380 GO SUB 1200: PRINT "Node ";
o
  390 LET h=0: LET vv=0: FOR i=1
TO v: LET r(i,o)=0: LET m=0: LET
 n1=0: FOR j=1 TO q: LET m=m+s(i
,j,o)*c(i,o)*q(j,o): LET n1=n1+q
(j,o): NEXT j: IF n1<>0 THEN LET
 m=m/n1
  400 FOR j=1 TO q: LET r(i,o)=r(
i,o)+ABS (m(1,i,o)-m(2,i,o))*(AB
S (s(i,j,o)-m)↑2)*c(i,o)*q(j,o):
 NEXT j: IF r(i,o)>h THEN LET vv
=i: LET h=r(i,o)
  410 NEXT i: IF h=0 THEN GO TO 5
50
  415 LET c(vv,o)=0
  420 FOR j=1 TO q-1: FOR k=j+1 T
O q: IF s(vv,j,o)<>s(vv,k,o) AND
 q(j,o)<>0 AND q(k,o)<>0 THEN GO
 TO 440
```

```
 430 NEXT k: NEXT j: GO TO 390
 440 INPUT (vv);" ";(v$(vv,o));
v(vv,o): IF v(vv,o)<m(1,vv,o) OR
 v(vv,o)>m(2,vv,o) THEN PRINT ''
"Input is out of range";"You gav
e minimum ";(m(1,vv,o));"and max
imum ";(m(2,vv,o)): GO TO 440
 450 FOR p=1 TO n: FOR u=1 TO a(
p): IF v$(u,p)=v$(vv,o) THEN LET
 v(u,p)=v(vv,o): LET c(u,p)=0
 460 NEXT u: NEXT p: LET d=mi: F
OR j=1 TO q: LET d(j)=d(j)+v(vv,
o)*s(vv,j,o): LET p(j)=d(j): IF
d(j)>=d THEN LET d=d(j): LET hj=
j
 470 NEXT j
 480 FOR i=1 TO v: FOR j=1 TO q
 490 IF c(i,o)=1 THEN IF s(i,j,o
)>s(i,hj,o) THEN LET p(j)=p(j)+(
s(i,j,o)-s(i,hj,o))*m(2,i,o)
 500 IF c(i,o)=1 THEN IF s(i,j,o
)<s(i,hj,o) THEN LET p(j)=p(j)+(
s(i,j,o)-s(i,hj,o))*m(1,i,o)
 510 NEXT j: NEXT i
 520 LET h2=mi: FOR j=1 TO q: IF
 p(j)>=h2 THEN LET h2=p(j): LET
hi=j
 530 IF p(j)<p(hj) AND j<>hj THE
N LET q(j,o)=0
 540 NEXT j: IF p(hi)<>p(hj) THE
N GO TO 390
 550 IF hj<>0 AND t$="y" THEN IN
PUT "Can I deduce outcome:";(q$(
hj,o));a$: IF a$="y" THEN GO SUB
 770: GO SUB 840: NEXT o: INPUT
"Continue Training?";b$: IF b$="
y" THEN GO TO 350
 560 IF t$="n" THEN PRINT "I sug
gest ";(q$(hj,o));"as likely.":
GO SUB 770: NEXT o: INPUT "Do yo
u wish to continue?";b$: IF b$="
y" THEN GO TO 350
 570 IF (a$="y" AND b$="n" AND t
$="y") OR (b$="n" AND t$="n") TH
EN GO TO 20
 590 FOR i=1 TO q: PRINT i;" ";
(q$(i,o)): NEXT i: INPUT "Which
outcome is it?";hi
 600 FOR i=1 TO v: IF c(i,o)>0 T
HEN INPUT "What value was ";(v$(
i,o));" ?";v(i,o): LET c(i,o)=0
 610 NEXT i
 620 FOR i=1 TO q: IF d(i)>=d(hi
) AND i<>hi THEN FOR j=1 TO v: L
ET s(j,i,o)=s(j,i,o)-v(j,o): NEX
T j
 630 NEXT i
 640 FOR j=1 TO v: LET s(j,hi,o)
=s(j,hi,o)+v(j,o): NEXT j: LET h
```



```
j=hi: GO SUB 770: GO SUB 840: NE
XT o
 650 INPUT "Continue training?";
c$: IF c$="y" THEN GO TO 330
 660 GO TO 20
 670 FOR o=1 TO n: GO SUB 1200:
PRINT "Node ";o;''"Input some exam
ples";'';"How many examples at th
is node?": INPUT n2
 680 FOR i=1 TO n2
 690 GO SUB 1200: PRINT "Example
 number ";(i);"on node ";(o);''
 700 FOR j=1 TO a(o)
 710 INPUT "Variable ";(j);"(";(
v$(j,o));") is";e(j,i+z(o),o)
 720 NEXT j
 730 PRINT '';"And the outcome is
?": FOR j=1 TO u(o): PRINT j;".
";q$(j,o): NEXT j: INPUT "Please
 Input by Number";e(a(o)+1,i+z(o
),o)
 740 NEXT i
 750 LET z(o)=z(o)+n2
 760 NEXT o: GO TO 20
 770 FOR p=1 TO n
 780 FOR y=1 TO a(p)
 790 IF v$(y,p)=q$(hj,o) THEN LE
T v(y,p)=m(2,y,p): LET c(y,p)=0
 800 FOR r=1 TO q: IF v$(y,p)=q$
(r,o) AND r<>hj THEN LET c(y,p)=
0
 810 NEXT r
 820 NEXT y: NEXT p
 830 RETURN
 840 INPUT "Keep this example?";
e$: IF e$="n" THEN RETURN
 850 FOR j=1 TO a(o): IF c(j,o)=
0 THEN GO TO 890
 860 LET vv=j: LET c(j,o)=0
 870 INPUT "What value was:";(v$
(vv,o));v(vv,o): FOR p=1 TO n: F
OR q=1 TO a(p): IF v$(q,p)=v$(vv
,o) THEN LET v(q,p)=v(vv,o): LET
 c(q,p)=0
 880 NEXT q: NEXT p
 890 NEXT j
 900 LET z(o)=z(o)+1
 910 FOR j=1 TO a(o)
 920 LET e(j,z(o),o)=v(j,o)
 930 NEXT j
 950 LET e(a(o)+1,z(o),o)=hj
 960 RETURN
 970 GO SUB 1200: PRINT '"Expert
 is working on it": FOR o=1 TO n
: LET de=z(o): LET i=0: FOR p=1
TO de*200: LET i=INT (1+de*RND)
 980 LET d=0: LET c=0: LET v=a(o
): LET q=u(o)
 990 FOR j=1 TO v
1000 LET d=d+e(j,i,o)*s(j,e(v+1,
i,o),o)
```

```
1010 NEXT j
1020 FOR k=1 TO q
1030 LET d2=0
1040 IF k<>e(v+1,i,o) THEN FOR j
=1 TO v: LET d2=d2+e(j,i,o)*s(j,
k,o): NEXT j
1050 IF d2>=d AND k<>e(v+1,i,o)
THEN LET c=c+1: FOR j=1 TO v: LE
T s(j,k,o)=s(j,k,o)-e(j,i,o): NE
XT j
1060 IF c=1 THEN FOR j=1 TO v: L
ET s(j,e(v+1,i,o),o)=s(j,e(v+1,i
,o),o)+e(j,i,o): NEXT j: LET c=c
+1
1070 NEXT k
1080 NEXT p
1090 NEXT o: GO TO 20
1100 GO SUB 1200: PRINT ''"Save
the current expert""(i.e. both
the program and data"''"Please i
sert cassette"
1110 SAVE "multinode"
1115 PRINT '"Saved under name 'm
ultinode'"
1116 PRINT "Press any key to con
tinue"
1117 LET a$=INKEY$: IF a$="" THE
N GO TO 1117
1118 GO TO 20
1120 CLS : FOR x=1 TO n
1130 PRINT "Node ";x
1140 PRINT "e(i,j,node)"
1150 FOR y=1 TO a(x)+1: FOR z=1
TO z(x): PRINT e(y,z,x);" ";: NE
XT z: PRINT : NEXT y
1155 PRINT "Press any key to con
tinue"
1156 LET a$=INKEY$: IF a$="" THE
N GO TO 1156
1160 CLS : PRINT "r(i,j,node)"
1170 FOR y=1 TO a(x): FOR z=1 TO
 v(x): PRINT s(y,z,x);" ";: NEXT
 z: PRINT : NEXT y
1175 PRINT "Press any key to con
tinue"
1176 LET a$=INKEY$: IF a$="" THE
N GO TO 1176
1180 NEXT x
1190 PRINT "Press any key to con
tinue"
1196 LET a$=INKEY$: IF a$="" THE
N GO TO 1196
1198 GO TO 20
1200 CLS : PRINT ,"EXPERT"',"___
____"'': RETURN
```


7.8 Some examples

If you actually managed to key in that program then the first thing you will want to do is to go off somewhere quietly by yourself and nurse your bleeding fingers for a week or so.

But, having done that, you might like to try running it - in which case some examples could be handy so that you know if it's working or not.

So, first, RUN and you should get a menu of seven items. Choose the first option to initialise the thing. Tell it you have one node. Tell it the maximum number of variables and outcomes is two.

At this node you have two variables. Name them as Wings (with minimum value 0 and maximum value 1) and Engine (minimum 0, maximum 1). Tell it you have two outcomes. Name them as Bird and Plane.

Now you should get the menu back and you can do one of two things. Either, go to a training session and let it learn slowly; or, go to the Input Examples option to give it a slab of examples to work from. We go to the Input Examples to make things simple. Give it two examples. It will ask about the variables and the outcomes so, for the first example, give it a bird and, for the second example, give it a plane.

Then go to Exercise Expert and put your feet up for a minute or two while it works out its rules for you.

When it comes back go to option seven to have a look at the rules it has developed. You should get array E holding the examples you gave it as follows:

1	1	-this line is Wings, both Yes
0	1	-this line is Engine, only on the second example
1	2	-this line is Outcomes, 1 for Bird, 2 for Plane

And you should have array R as follows:

1	-1	-this line for Wings
-2	2	-this line for Engine

Have a look at these rules and satisfy yourself that they will separate the two outcomes the way they should. Once you see that they do you will feel easier about the prospect of the program as a whole working - although there isn't really any need to do this every time.

Press any key to return to the menu.

Go to option 5 - Normal Running.

The program will first ask about Engine. Notice that in array R this seems the most important variable (which it really is!). Reply 1 and the expert will guess Plane without asking anything else.

Reply Y to continue.

It asks about Engine again. Reply 0. It now asks about Wings, reply 1. It guesses, correctly, that it is a bird.

If it does all of this the program is working pretty well and you can try something rather more adventurous.

Try two nodes. On the first node set it up to establish whether the object you have in mind is a machine or an animal.

For instance:

Variables	*Outcomes*
Feathers	Animal
Metal	Machine

On the second node set it up to establish whether or not the object is a bird or a plane.

For instance:

Variables	*Outcomes*
Feathers	Bird
Metal	Plane
Animal	
Machine	

When you input examples on node one, think first of an animal and answer the questions with this in mind. Then think of a machine and answer the questions with that in mind. Two examples in all.

Then give two examples on node two. One with a bird in mind and another with a plane in mind.

Take particular care to spell the words the same everytime you key them in or the string matching will go wrong.

Then exercise the expert.

When it comes back send it off into normal running.

The first thing it asks about is Feathers. Suppose we were thinking of a bird and reply 1. It then suggests Animal as likely and asks about Metal on node two. Reply 0 and it suggests Bird as likely.

What happened was that the one variable, Feathers, was enough to settle node one and enable it to deduce that the object was an animal. Now, if we look at node two we see that Feathers also occurs there - so it knows about that - and that it had already deduced a value for Animal and for Machine (1 and 0, respectively) so that the only point it was not sure about was Metal. So it asked about that, got a reply, and could then solve node two correctly to guess Bird.

Give it another try, thinking of a Plane.

Feathers? Reply 0. It deduces a Machine and, again, asks about Metal. Reply 1 and it deduces that it is a plane.

Hopefully, by now your aching fingers are recovering and you feel a bit better about the effort involved.

And the question is: Can it now do anything useful? Something other than an apparently trivial game. Well, we can give it a try with a real life problem - like trying to diagnose faults on a cassette recorder.

We'll suppose that there are the following faults which you could observe on your recorder:

1. No lights
2. Tape won't move
3. Unit won't record
4. Intermittent sound
5. Distorted sound
6. Erractic speed
7. Hum

These could arise from any combination of the following causes:

1. Not switched on
2. Deck in "Pause"
3. Tape jammed
4. Tape inserted wrongly

5. Erase tab removed
6. Dirty head
7. Stretched tape
8. Poor recording
9. Amplifier problem
10. Dirty capstan
11. Wrong leads

And the remedial action could be anyone of the following:

1. Switch on power
2. Press "Pause"
3. Replace cassette
4. Re-insert cassette
5. Clean heads
6. Re-record tape
7. Check amplifier
8. Clean capstan
9. Check leads

So set up a two node system. The first node is to find out what is wrong so it has as its variables the seven faults and, as its outcomes, the 11 causes.

The second node is to suggest remedial action so it has the 11 causes as its variables and, as its outcomes, the 9 remedial actions.

Input examples.

For the first node give 11 examples, one of each cause, and answer the system's questions with the specific cause in mind. For the second node give 9 examples, one for each remedial action, and answer questions on the causes in such a way that you reply "1" for each cause that might be helped by the remedial action and "0" for each cause that would not be helped by it.

Then exercise the expert (or, alternatively, instead of inputting examples and exercising, you could have had a training session instead). It's only fair to point out that, as the system gets bigger, the training and exercising will take longer - so don't think the computer has gone and died if it goes away for a while to work on the problem.

When it comes back, having worked out a rule set, you can send it to normal running when it will try to figure out what's wrong with your cassette recorder.

Exactly how it runs will depend on the examples you gave it to work on. It may also depend to some extent on the random number generator in your computer if you 'trained' it using Option 3 "Exercise Expert" - this is because the actual rules formed can depend on the order in which the examples are presented and not all random number generators produce the same stream of random numbers. But, by way of illustration, this is what happened during one session:

It asked if there were No Lights. Replying 1 (yes, No Lights) it suggested Not Switched On as likely and then passed straight to node 2 when it immediately suggested Switch On Power.

The next session we decided that the problem was Intermittent Sound.

It asked 'No Lights?' - reply 0

It asked 'Intermittent Sound?' - reply 1

It asked 'Tape Won't Move?' - reply 0

It asked 'Erratic Speed?' - reply 0

It asked 'Unit Won't Record?' - reply 0

It asked 'Hum?' - reply 0

It asked 'Distorted Sound?' - reply 0

It then deduced that Stretched Tape was the most likely cause and, turning to node 2, it advised Replace Cassette without asking any further questions. All of which seems pretty reasonable.

Now, trying it on Tape Won't Move it kicked off by asking three questions to which we replied '0' and, then, when it asked 'Tape Won't Move?' we reply 1. It then asked one more question, to which it got a '0' reply, after which it suggested Tape Inserted Wrongly and, moving to node 2, it advised Re-Insert Cassette. Not bad. Not bad at all.

Give it a try yourself, these are the examples it was trained on:

Node 1 (Diagnosis)

Variables (Faults)	Outcomes (Causes)										
	1	2	3	4	5	6	7	8	9	10	11
1	1	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	0	0	0	0	0	0	0
3	1	1	1	1	1	0	0	0	0	0	0
4	0	0	0	0	0	1	1	1	0	0	0
5	0	0	0	0	0	1	1	1	1	1	1
6	0	0	0	0	0	0	1	0	0	1	0
7	0	0	0	0	0	0	0	0	1	0	1

Node 2 (Remedial Action)

Variables (Causes)	Outcomes (Remedies)								
	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0
5	0	0	1	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0
9	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	0	1	0
11	0	0	0	0	0	0	0	0	1

Once you have this up and running it soon begins to seem possible that something useful could actually be done with this system - after all, the program as it stands would take a lot more nodes, a lot more variables, and a much more complicated set of interconnections for it all.

You may well feel that it would be more natural, in many of the examples given so far, to reply Yes or No rather than 0 and 1 - and you're probably right. In fact, you could tailor the program quite extensively to behave in a more natural way for a given subject. But just a brief reminder (again): the input variables do not need to be all ones and noughts.

They could be any real numbers - like rainfall figures, for example. Bear this in mind when you are developing an idea for the expert system and it could help to improve the performance.

Note: On the Apple II and many other micros, all variables and arrays are initialised to zero when you key RUN. If your micro does not do this then you will need to clear all the arrays and some of the variables at the start of each run - setting them all to zero in the program code, right at the beginning. Also, some micros (including the Spectrum) have to be told in advance if any variable is assumed to be zero or null, as in a statement such as

```
LET C =Ø
```

some micros assume C to be zero if it's not been mentioned before. The Spectrum requires you to set it to zero.

Chapter 8

How can you use your Expert?

8.1 Choosing a problem

Suppose that now you want to build your very own expert system. You don't want to build the system described earlier in this book, you want to build something which is peculiarly yours. Well, the first thing to decide is: About what shall this system be expert?

You can, if you like, reply that it has to be expert about absolutely everything in the entire known Universe. There's no harm in such an aspiration. But, if you do, you'll be likely to come up with something that looks a bit like the system described in the previous chapter. After all, to be expert in everything you need a very general design with an absolute minimum of preconceptions. The snag with this approach is that, whilst it might work on a wide range of problems, it might not be outstandingly good at any of them in particular.

Ideally, you might choose an area of intended expertise which is not too broad - and not too narrow. Which sounds a bit vague (and is) but has some reason behind it.

Suppose that you choose too narrow a field to work in. Like, for instance, diagnosing a fault in a motor car. Now you might think that this field would be fine - after all, motor mechanics charge a lot and to get a computer to take their place could be handy. But look at the problem more closely.

Say your car won't start in the morning. What you want is for the computer to diagnose what's wrong with it. So: look in the owners' handbook and you'll probably find that it won't start for one or more of the following reasons. It might be out of petrol; have a flat battery; have water in the

distributor; or, have dirty spark plugs. So put that on an expert system. Now run (mentally, as it were) that system. And, on the screen is the question: Is there petrol in the tank? And you trudge out to the car to have a look and get an answer. If there was, you might be asked: Is there water in the distributor? And, again, you trudge out to look. And, if there isn't, five minutes later you're out there unscrewing a spark plug to see if it's dirty...

And, all in all, you really didn't need to switch the computer on just for that. You could have just taken the owner's manual out to the car with you and saved on electricity.

The reason is that diagnosing why a car won't start is very trivial. The real effort is in poking around the car trying to get the information you need in the first place.

The computer can't poke around the car for you in this case - so what you really need is a car mechanic to help you.

Certainly, there might be some tasks in car mechanics which could benefit from an expert system. Suppose that you ran a workshop and regularly had relatively inexperienced mechanics checking out different vehicles and carrying out (fairly) complex work on them. Then you might think it worthwhile to have a screen set up to advise them on how to proceed. Alternatively, you could just give them the workshop manual. But, if it's simply a matter of a car that won't start, that really does seem like a problem which is much too narrow to be worth tackling. The problem has to be 'large' enough to actually give the computer some useful work to do. And it will be most useful if the task isn't already accomplished in some other medium - such as a workshop manual.

At the other end of the scale the problem shouldn't be too large. The reasons for this are practical ones again - primarily, the practical problem of getting an expert system built with enough expertise in a large area. As the size of the problem area increases so does the amount of effort on your part necessary to carry out a thorough implementation. And, if the implementation isn't thorough, the usefulness of the system is pretty dubious.

Real Life Expert Systems aren't expert in, for example, the whole field of medical diagnoses. They are experts in a narrow field - and are good in that field. An all-singing, all-dancing system that frequently makes the wrong diagnosis is liable to kill as many patients as it cures.

Finally, the system will have the most chance of being useful if there appears to be some method of getting it to work. This sounds rather silly

but suppose you had a system which you intended to be expert in the field of winning the football pools. Now how, even roughly, would you produce such a system? Some sports and games of chance might be amenable to a bit of computerised prediction. But football? All you need is a few key players to stagger onto the pitch with hangovers and the result of the game can be very different from any result the computer might dream up. The problem is too diffuse to be handled by computer. It isn't at all clear what sort of rules govern the actual play of the game and it isn't at all clear that there's any method by which you could uncover any rules.

In general, you can get an idea as to whether you have a likely field for an expert system (or any program) by asking yourself whether or not there's anything much in the problem that can be measured.

If there is, then you're in with a chance. If not, forget it.

By 'measured', of course, one doesn't necessarily mean length, breadth and width. A yes/no response is a measure of sorts which will keep a computer happy. But, overall, you must be able to describe the problem area in terms of a series of measures of some sort. If there is something in the situation that can't really be reduced to a measurable quantity (like the fine footwork of player X) then you have something which is unlikely to be convertible to computer.

8.2 Analysing the problem

Once you've hit upon a problem area which looks worthwhile the next thing to do is to start analysing it.

First, you need to get a broad overview. Typically, you already have this. After all, most systems don't get written by accident. They tend to arise because a specialist in some field thinks a computer could help or because a computer person knows a specialist with a problem. If you don't have this overview then this is the time for gently poking around in the area - primarily to see if you were right to think that this is something which could be done on a computer. To check that it really was a suitable problem to choose.

And then, the typical next step - you grab an expert and pick his brains on the subject. And you can divide this up into a fairly orderly sequence consisting of the outcomes, the measurable evidence (variables) and the reasoning that links them.

The outcomes may be very simple - gold in them thar hills, bronchitis in

them thar lungs. That sort of thing. Is it, though, the presence or absence of an outcome which is important? Or is there some other measure associated with the outcome? A probability, for instance. It's important to sit down and work out what, ideally, you want to get out of the system and in what form you want it. A list of possible diagnoses, or conclusions, or recommendations and an indication of how these results are to be measured.

The variables are the pieces of evidence that the human expert has at his disposal. You have to find out what they all are and how they are measured. Having asked the expert what he hopes to find out you need to know what he considered in coming to his conclusions.

The linkages between these items are the rules the expert applied. What 'internal program' was the expert working through when he came to his conclusions? This can easily be the hardest bit. Quite possibly, the human expert won't be fully aware of what rules he uses. So all you can do is to get a first crack at finding out what he thinks he does and then go away and implement it to see what happens.

Once you've got all this initial information together you may find that a form for the program naturally presents itself.

For instance, with our totally general-purpose system the requirement of generality of purpose dictated that it had to be self-learning, producing its own rules from examples.

It also became evident that the particular inferencing structure used wasn't too critical - because what might be right for one application might be of less use for another (so, we produce a system that's only of marginal use in any situation...)

But if you have a known list of outcomes and a known list of variables and a known list of rules in front of you - then the situation is different. If the rules are of very widely varying type then you might have to write specific code for each one. If they can be reduced to a common format you can store them on file as if they were data and then write a routine for working through them. If they all rely on logical connectives you can write fairly simple deductive code to arrive at definite conclusions. If they have probabilistic elements associated with them you need some method of keeping track of the probabilities.

It seems a bit feeble to say that one can't advise in great detail on what you should do - that you should work it out for yourself. But, in fact, this actually is what you have to do. You can look at some of the ideas in this

book and they may give you ideas to help in working out an approach. But just as a general-purpose expert system falls down on fine detail so will a general-purpose method of building an expert system. Each application area will have its own peculiarities which suggest a special treatment.

People who have built expert systems of their own frequently report that the hardest stage was the initial accumulation of outcomes, variables and rules. Once they had derived this information from a human expert and set this information down on paper the rest began to fall into place before them. But, even if no absolute structure appears at this stage there is a further reported fact which makes things slightly easier. Namely, that the process doesn't stop there.

With the initial information you set something up to run on the computer - a tentative sort of program which you think might work. You then run it on a few examples and hand the results to the human expert for comment. Usually, the system makes mistakes and between computer person, human expert, and computer a process of feedback sets in wherein the program rules are altered progressively until the thing starts working reasonably.

In some ways you could feel that the process is pretty sloppy. After all, if you set out to write a payroll program without much idea as to how payrolls were calculated then people might feel you were in the wrong business and should try, say, chicken farming instead.

But, this is how expert systems seem to be developed. And one could argue that it's a reasonable method in a situation in which nobody really knows exactly how the thing should be done. After all, if it eventually works the means used probably justify the end.

And one has to admit that, in fact, one does know people who have written even such things as payroll programs by a method not entirely different from this and have managed to sell the end product.

The really hard part of the problem to analyse, actually, is why any human expert should spend his or her, apparently valuable, time handing away the secrets of their art to someone who's going to put it on disc, make a million copies, and sell them to anybody at just £5 a time. Such an expert must, surely, be mad. It's as if one of two things were likely to happen: One, that the system won't embody their expertise, will be fairly useless, and therefore no harm is done to their business; or, Two, that they really don't mind having their expertise devalued and being put out of business. After all, where have all the payroll clerks gone? Perhaps they're standing

in the dole queue wishing they could meet that nice computer person one more time?

Maybe you should be generous and offer your human expert a cut of the royalties on your system. Maybe you should be generous and offer me a cut of the royalties, too.

Chapter 9

Large-Scale Expert Systems

9.1 MYCIN - medical diagnoses

So far, all we've considered is the one expert system - our very own, totally general-purpose, home-made expert system. There are, of course, others. To what extent does our system resemble existing systems? Well, the best answer is to describe a few others and see just where the similarities, if any, lie.

MYCIN is an expert system designed to carry out medical diagnoses. Specifically, it's designed to work in the area of blood and meningitis infections - making an appropriate diagnosis from evidence presented to it and recommending a course of drug treatment for any diagnosed infections. It consists of a total of 450 rules developed with the help of the Infectious Diseases Group at Stanford.

Its most fundamental point - and the one which can give rise to the most complications - is the use of probabilities. Medical diagnosis is an inexact science. If a patient exhibits a particular set of symptoms then they might well indicate a particular illness, but the connection is rarely total. Consider the contrast between a medical diagnosis system and a system which was expert in, say, the field of chemistry. To make it easy, let's consider hypothetical systems - because, apart from anything else, this helps to avoid tresspassing on the preserve of real, human, experts.

Suppose I have an expert system for chemical analysis and one of the pieces of information I give it is the result of a litmus paper test. That is: on adding litmus paper to the solution in question it goes, for instance, red. Now, from this the expert can 'diagnose' that what I have is acidic. Easy. There are no doubts present.

Now switch on the medical diagnosis system and inform it that the patient under consideration has a bad cough. Well, that might mean that there is a

case of bronchitis, tuberculosis, or, well... just a bad cough. There is no absolute certainty about the meaning of the evidence.

And, somehow or another, the expert system has to be able to cope with this uncertainty. Our own system did this to some extent - but not very precisely. And, if you recall the earlier discussion on probabilities you'll recall just how difficult it is to deal with a problem like this.

The way MYCIN tackles the problem is to assign a Certainty Factor to every one of its 450 rules. So you can think of MYCIN as containing a series of rules of the form IF...THEN with certainty P.

And now, note that we used the phrase 'Certainty Factor' rather than the word 'Probability' and the question is: Why? Are they different?

Well, you know all about probabilities by now if you've read the earlier sections of this book but what you didn't realise is that there can be more than one type of probability. The type we've looked at so far (and which we'll continue to look at) are statistical probabilities. The whole theory of statistical probability is based on the assumption that, if only you had enough examples, these statistics would accurately describe the behaviour or the system you're looking at. It is the frequency approach to probability.

Some people, however, maintain that this is the wrong approach to use for inferencing systems - systems which modify their degree of belief in an outcome depending on the inputs they receive. For these systems, it is claimed, a theory of Logical Probability is better than a theory of Statistical Probability because, in the case of an inferencing system, there isn't really an external frequency model for what is happening. To go into the detail of Logical Probability is somewhat beyond the scope of this book but, by way of slight compensation, this author would suggest that if you stick to a statistical model then you won't, really, go far wrong and that the theorists of Logical Probability are not, in fact, particularly good at pinning down the details of the calculations you should make, even if you did accept their theories (this criticism of Logical Probability is known as the Sour Grapes Theory). In other words, even if you knew the theory backwards it probably wouldn't improve your program much.

Anyway, Logical Probability is the approach used by MYCIN and the practical effect is that MYCIN's Certainty Factors are, roughly, what most people would think of as Conditional Probabilities of the form P(H:E) - the probability of this hypothesis given this evidence. Because they aren't really probabilities in the sense we've discussed them the calculations which we've used so far don't really apply - and MYCIN uses a fairly *ad hoc*

method of summing up its certainty factors as it proceeds through the program.

So, to start at the beginning, where did these Certainty Factors come from? In the case of MYCIN they came from the human experts who provided the rules in the first place. When they suggested a rule they stated their degree of confidence in that rule on a scale from 1 to 10.

And the point we made earlier concerning probabilities arises - how do we know that these probabilities are correct? Well, we don't really. Doubtless they're fairly correct (and the fact that MYCIN gives good results supports the suggestion that they are pretty good) but the method is essentially *ad hoc* - which means it wouldn't make a statistician happy.

Having set up these rules with their associated certainties MYCIN works by backward chaining from a possible outcome to see if this outcome can be believed or not. Once it's established all of the items it needs concerning a particular outcome it makes a judgement on that outcome calculated on the basis of the certainty factors associated with all of the rules which had to be used to reach that particular outcome.

For instance, if the outcome were item Z, it might have been necessary to establish both X and Y in order to deduce Z. But the rules used to establish X and Y might have certainties P and Q associated with them. Now if P and Q were each of value 1.0 say, then Z would necessarily follow. If P and Q are less than 1.0 (which, in general, they are) then Z doesn't necessarily follow. It only follows with a certain amount of certainty.

And, recalling the earlier discussion of probabilities, it is no mean task to calculate just what the certainty of Z will be under these circumstances. So much depends on the exact form of all of the items concerned and how they interrelate.

So MYCIN, instead of trying to get an exact solution to this problem, simply cumulates all of the certainties concerned to give an idea of the sort of relative magnitude of the answers.

Probably the important point to note is that MYCIN doesn't come up with a diagnosis and disclose the exact certainty of that particular diagnosis being true. What it does is to come up with a whole series of diagnoses each of which has some kind of certainty 'score' associated with it. Above a certain - *ad hoc* - value all of these diagnoses are accepted as being, to some extent, likely and the user is presented with a list of possibilities.

Mathematically the procedure is somewhat shaky but, against that, the

evidence is that it works very well in practice. So much, of course, for mathematics...

Medical diagnosis, though, is itself something of a shaky procedure. A doctor doesn't give an exact probability statement about each of his patients - he simply reckons that a particular diagnosis seems kind of likely with, maybe, some other diagnoses being additional possibilities. Further, a patient might not be suffering from just one complaint - he might be suffering from several things simultaneously. In this case, to work out the one, best, exact, diagnosis is to exclude the perfectly valid possibility that there just isn't one best, exact, diagnosis, but several.

It has been reported that members of the medical profession, encountering MYCIN, have been quite happy with it reckoning its abilities as good as theirs. Possibly that is where the real proof of the pudding actually lies.

The next point to make about MYCIN is its use of the English language. When it wants information off the user it asks for it in an English language way. When the user enters information he does so in a way which appears fairly natural and English-like. This sounds handy - but is it anything more than that?

There are two parts to the matter - the part that concerns user acceptance of the finished product (was it worth doing?) and the part that concerns the actual implementation (was it easy to do?).

Taking the first part it's reckoned that one of the advantages of expert systems is that anyone can use them with very little previous knowledge of computers. Certainly, it's easy to see that users would respond more favourably to a system that used their language than a system which forced them to talk in, say, BASIC. Doctors, for instance, using MYCIN found the language easy to work with and, conceivably, wouldn't have wanted to waste their time evaluating a system that was hard to use. And, if nobody's willing to waste their time using a system then one might argue that its long-term usefulness was pretty non-existent.

But what about implementation? After all, everyone knows that natural language processing is one of the most difficult jobs around - how do you get around the problem?

The answer, really, is a bit of a cheat because, in fact, MYCIN doesn't really carry out full natural language processing at all.

The trick is that every profession tends to play its own little language

game. It has special words, stereotyped ways of saying things, that are quite particular to that profession. There are lots of reasons for this, some of them good reasons. At the semi-malicious level one could point to the fact that even small children often have their own private language which they use to exclude outsiders from their conversations and which increase solidarity amongst their friends. Adults have these private languages too (including computer people - they're the worst of the lot). At a more significant level, when one is talking about precise concepts one has to use words in a precise way and in the same way every time you use those words. This leads to a specialised subject language which might look the same as normal English to an outsider - but which is actually very different.

Take for instance the words 'chronic' and 'acute'. In medical parlance these words simply refer to the duration of an illness - whether it has been around for a long time (chronic) or only a short time (acute). So, clearly, when a doctor says his patient has a chronic cough he doesn't mean that it's simply awful. Nor, if he says that a cough is acute, that he thinks the patient is going to drop dead any minute. He's simply making a statement about the length of time the patient has been coughing like that.

All of this might seem like an unnecessary diversion - but it can help a lot in implementing an expert system. For MYCIN it was found that doctors working in this area of diagnosis used words in very precise ways and uttered very stereotyped comments. Much more so than most people would do in the course of a normal conversation. And the advantage of this was that it was possible to easily define a very limited subset of English which would express everything that might be said on the subject with very few complications. Standard phrases and forms of grammar were readily adapted into the program and the result was a highly stunted subset of English which was easy to program.

The doctors were happy with the result because, possibly without realising it, they too spoke in a highly stunted subset of English. At least when they were discussing their work they did. For all one knows they turned into famous orators once they got home of an evening - but that is beside the point.

In a way this all ties in with the comments on DENDRAL which, you'll see later, doesn't use any English language at all. What it uses is a graph language suited to the particular activities of chemists. That is a very restricted subset of English - but the point is similar. Model the system into the language of the human experts who will use it and you increase its chances of user acceptance and can more readily take advantage of the knowledge that already exists in that field which will, to some extent, have shaped the language used to describe that knowledge.

The bad news though (there's always some bad news) is that, having done this, you have an expert system which is more difficult to adapt to other areas of expertise, simply because of the differences that exist in the language used to describe expertise in other fields.

On the subject of user-acceptance MYCIN has a capability which many other expert systems possess and which is frequently commented upon - its ability to explain to the user why it is doing what it is doing.

As a simple example, suppose that you have an expert system and it asks you if you have a cough. Instead of simply replying yes or no you could ask the expert: Why? That is, why has it asked that question. The system could then display a message pointing out that a cough sometimes indicates lung trouble. Or some such comment.

Now, at its very simplest level, programmers will realise that this is nothing more than a simple program comment. Every time a rule is programmed into the system all you need to do is to program in a brief piece of text giving an explanation of the purpose of that rule. So, when the system asks a question involving that rule it can invoke this piece of text as an explanation if the user wants it. It resembles the REM, in BASIC.

In other words, it's not very clever. So why the interest in it?

In the finished product, the reason for doing this is simply user acceptance. Users, particularly those who don't know much about computers, are impressed with a system that acts in such a human fashion that it can explain itself when asked to. And, as noted before, there's not much point in writing a system that nobody's going to use.

But there's rather more to it than that. Consider the programmer (you, for instance, writing the system). Program REMs are all well and good. They certainly assist in debugging and design because they remind you of why it was you included that bit of code and of what it is supposed to do. But the snag with REMs is that you have to list the program in order to read them. It sounds rather a trivial complaint but in a long program which might have finished up almost anywhere there might be a better way of recording what's going on.

And that's, to some extent, what the Why? of the expert systems represents. They are 'live' REMs that can be called at any time and remind you of what's going on without actually interrupting anything else.

Now stand back a little further from the actual program and recall the problems that are usually encountered in building up an expert system -

the problem of knowledge acquisition and engineering. The program contains a series of rules and reasons for those rules and, typically, the program doesn't work perfectly as yet. The human expert is sitting at the screen working through an example. Suddenly, as it were, the machine asks a stupid question. At this point the (human) expert can now ask Why? and get some idea of what's gone wrong with a view to putting it right by adding another rule or modifying some old rules.

All of which would not be very earth-shattering if, really, all the system did was to print out a standard comment. Something a bit more sophisticated would certainly help.

In the case of MYCIN this something-more-sophisticated is TEIRESIAS - a system for modifying the MYCIN rule set and explaining the actions of MYCIN. In essence it's very much like a trace and dump facility with the big advantage that it's somewhat more user-friendly than the traditional technique of filling a box of lineprinter paper with the contents of main memory in hexadecimal.

With our human expert sitting at the MYCIN screen it's possible to ask *why?* in response to a request for information and receive a summary of the line of reasoning that has been followed so far. Specifically, it can display the current rule that has asked for information and show the status of all the other inputs (if any) of that rule.

If you think of the system in the early part of this book you might have given it the problem of identifying an object as either a bird, a plane or a glider. The system, at some point, asks you if the object has an engine and you ask: why?

Obviously, it would be fairly easy to check through the current node to see what the state of the other variables was. In which case the system might point out that it has already determined that the object has wings, and doesn't have a beak, and (with an extra piece of coding) it could calculate that if it got a Yes response to Engine then it could conclude the object was Plane but a No response would cause it to conclude Glider.

And if the rules it was working from were faulty it might announce its intention of deducing Glider if the object had an Engine - which would be wrong, thereby letting the user know that this particular rule needed modifying.

Naturally, a system like MYCIN needs more code to enable it to tailor its statements, but the principle is the same. *Why?* provides a snapshot into the current reasoning position of the system which is a useful aid in initial

development and debugging as well as serving to reassure the eventual users that it isn't just working at random.

Another facility of TEIRESIAS is the question: How? applied to any given statement. Thinking again of our bird/plane/glider example one might have introduced code to enable the user to ask *how ? wings.* In other words, the system believes that the object in question had wings - how has it come to believe that. The answer would be simple in this instance - because the user told it so. And it would be simple to print a short message to that effect.

More complex systems need more complex methods though because, typically, the user wouldn't ask about the validity of some statement which he himself had made. He would be asking about some intermediate conclusion drawn by the system itself. In this case the technique is to step backwards through the chain of reasoning that leads up to this intermediate conclusion showing what rules were used and what information was used. The obvious application of this facility is again in program development at the point where the system has made a mistake - by asking an inappropriate question, for instance - and the human expert wants to know how it managed to get to where it now is. If Why? is the snapshot facility then How? is the trace.

It's fairly easy to see that enhancements like those offered by TEIRESIAS can be useful in developing an expert system but, even with a system that works perfectly, they have some use.

For, if the system can work well, and if the system includes the means to explain its actions in fairly English-like terms, then you have a system that could be used to teach others about its area of expertise.

After all, if you have a medical student who isn't very good at diagnosis and a computer system which is very good at it then you might as well sit the one down with the other and let them get on with it. A technique such as this could, one supposes, dramatically reduce the incidence of apoplectic fits amongst those who would formerly have been landed with the onerous task of educating the young.

This approach has, in fact, been tried.

A program, called GUIDON, has been developed to work with MYCIN in order to exploit MYCIN's knowledge about diagnosis for teaching purposes. And, again using GUIDON, the set of rules in PUFF (see next section) has been adapted to MYCIN so that teaching work can be carried out in the field of breathing disorders.

Obviously, it would have been possible to use these expert systems as they stand for teaching purposes. But some modification can enable the system to act as a more closely-involved monitor of the students' behaviour with a higher degree of interaction than would be possible if the student just sat there staring at the screen until it was time to go home.

With all the work that's been done in the field of expert systems for medical diagnosis you'd think that there was no need for doctors anymore, really. Doubtless there's some truth in this belief but as yet no medical authority has suggested that an expert system could be licensed to practice in its territory despite the reported comments that these systems are as good as the human experts. And (one supposes) if there's going to be a licensed (human) expert on hand one might as well save some money and get him to carry out the diagnosis as well.

A bit of a pity, really. The idea of an expert system with a pill dispenser just below the keyboard is rather a nice idea. Unless, of course, one happens to feel unwell oneself. That would then be a very different matter.

9.2 PUFF - breathing disorders

Having considered MYCIN, consider the following:

What would happen if one took MYCIN and shook it over the wastepaper basket until all of the domain-specific knowledge fell out leaving only the basic reasoning mechanism?

The answer is that you would have EMYCIN - Empty MYCIN - which would be a more-or-less general purpose expert system that, temporarily, wasn't expert in anything at all. So that's what the scientists at Stanford did. And, having got an Empty MYCIN they then proceeded to fill it up with something else.

That Something Else was a set of 50 or so rules concerning pulmonary disorders and, once put into EMYCIN, they gave rise to PUFF a, rather happily-named, program for diagnosing breathing disorders.

The idea is that a patient staggers into the doctor's surgery and breathes into a machine. There's nothing new in this - the machine simply records the volume of air the patient breathes and how fast the air moves when he breathes it. From this record a doctor can make some kind of diagnosis of the patient's condition.

For instance, he might be normal, he might be sick, the sickness might be bronchitis, it might be emphysema, it might be a number of things.

Whatever it is, the idea is to input data into PUFF and have PUFF work out a diagnosis.

For a start, it's worth noting that PUFF doesn't receive its data straight off the machine into which the patient breathes. Doubtless it could be modified so it did, but it doesn't.

What happens is that the machine presents the doctor with several, possibly relevant, pieces of information about the patient's breathing. The doctor also has to hand certain, possibly relevant, pieces of information about the patient in general. For instance: the patient's sex, age and smoking habits.

At this point you can forget about the breathing machine and turn your attention strictly to the computer, ready-loaded with its expert system. For all that we have now is a list of variables and certain values associated with those variables and the machine has to make a diagnosis.

As a trial run 150 sets of patient data were presented to PUFF to see how it got on - and the results were that PUFF made the same diagnoses as a human doctor about 90 per cent of the time.

Now, at this point, it's possible to see that this is the sort of thing which could be set up on our own expert system, described earlier.

Taking all of the variables to be considered we could have set these up and listed all the possible outcomes (perfect health, bronchitis, etc.). We could then have presented the system with the 150 sets of data in a training session and let the expert develop its own rules for forming diagnoses. We might then have found that it was, occasionally, right in making subsequent diagnoses.

However, it would be pretty unfair to the scientists at Stanford to suggest that the systems are identical in every respect because they aren't.

For a start PUFF doesn't work out its rules for itself. The Stanford scientists got together with others from the Pacific Medical Centre who actually told them how to make the diagnoses and, for most expert systems currently in use, this approach is far more usual. One reason for this could be that they don't like to trust the program to do too much by itself - but a more likely reason is that it's usually possible to build a more efficient system if you know, in advance, just how you want it to proceed.

So the medics came up with a set of rules by which diagnosis could be carried out. And the computer scientists implemented these rules in

PUFF. Put like that it sounds fairly easy and, in fact, Stanford makes no claims to the effect that it was hard.

About the form of these rules, it's worth making a few points. For a start, like MYCIN its rules are in the IF...THEN... format. So, we might have (in strictly non-medical language) IF (the patient can hardly breathe) AND (he smokes 200 ciggies a day) AND (he can't stop coughing) THEN (he has a smoker's cough).

Now this (apart from the specifics of the items in brackets) is how many workers describe their expert systems - as a set of IF...THEN rules. This way of describing things has one prime advantage - that anybody who uses computers knows what IF...THEN means. You could even write a BASIC program in which you actually code in terms of IF...THEN. (There is no disgrace at all in doing this - in fact, PUFF is one of the few expert systems that actually has been re-coded into BASIC.)

But it's as well to avoid the trap of thinking that this is exactly how it must be done. All the IF...THEN statement consists of is a proposition - a statement in logic. And there are many ways of making the same statement without using the words IF...THEN at all.

For instance, our expert system doesn't store its rules in IF...THEN terms but it holds exactly the same information as if it did. The expert generates its own rules and, every time it applies them, it effectively performs the same logic as if there were an IF...THEN statement there whose conditions correspond to the rules it has developed.

There's quite a range of terminology associated with the rules in expert systems. Commonly, they're called 'production rules' because they can produce an outcome, or conclusion. Often the first part of the statement, following the IF, is called the 'antecedent' and the second part, following the THEN, the 'consequent'. But you can also call them: fact and hypothesis, assertion and deduction, variables and outcomes, or whatever suits you. The only good reason for standardising on terminology is so that other people can understand you and, in a fairly new field, there isn't much standardisation of terms as yet to provide many guidelines as to what other people will understand.

In its early days PUFF had only 55 rules imbedded in it - which is quite encouraging, because most computers (one would think) can cram 55 rules into them and it's nice, therefore, to think that most computers can be made to contain something useful.

Now, before we go on to think about what PUFF does with its rules there's

a few more things to say. For a start, PUFF didn't work too well at first. A typical complaint of most programs under development, this should give the amateur cause for cheer. The problem was that the medics didn't supply perfect diagnostic rules in the first place. The rules they gave the people at Stanford simply weren't logically capable of diagnosing everything that came along. And this is where the Knowledge Engineer comes in.

PUFF produces a faulty diagnosis. The medics say it's a bad diagnosis. So (says the Knowledge Engineer) what should it have been? And then: why? Which of the existing rules was wrong? What new rule should be added?

So the medics think about it and come up with a few suggestions which are added into PUFF's rule base. It sounds a trivial procedure, this process of fiddling around with the rules until the thing works, and, intellectually, maybe it is. But it isn't a trivial problem as far as building an expert system is concerned. In fact, it's one of the most commonly-reported problems there is.

On the mechanical level, there has to be a facility for tinkering with the system to get it working better - an easy way of modifying and adding rules. This is because almost every system will need altering sometime and the danger is that if it's hard to do then it won't be done as often as it should be or as thoroughly. A practical point, but a worthwhile one.

On the more abstract level, you would think that a bunch of medics could have got their rules right in the first place. And, again, it's an important point to note that they couldn't. The human expert working on a problem usually has some idea of how he or she solves a problem - but most people are agreed that, initially, they don't have any very exact idea. Certainly not exact enough to get a computer to copy it straight off first time.

What happens is more like a process of mutual learning in which both the computer program and the human expert find out how the human expert has been working over the years. Possibly aided by the efforts of the Knowledge Engineer standing between the expert and the program, the program gets more sophisticated as the exact methods used by the expert are gradually untangled.

All of this points up the basic problem of how to get the expert knowledge into the machine in the first place. In many ways it's not strictly a computer problem. It's a problem of finding an expert and understanding what he's talking about sufficiently to be able to write a program to do what he can do. Rather more like systems analysis than anything else. But it's not a problem which can be ignored altogether because it is often regarded as

the biggest (smallest?) bottleneck in the whole business of building an expert system.

We cheated, of course. By designing a system which formed its own rules we sidestepped the problem of finding and understanding a human expert - but this may well have been done at the expense of the finished product.

The important point to note is that, apart from the simplest systems, it should be fairly easy to add new rules and modify old ones.

Exactly how the rules are held isn't important. They could be explicitly coded (with the hazard that this can make extensive alterations difficult) or they can be stored as an antecedent/consequent list as if they were data. This latter method is one which, theoretically, expert system builders favour. By treating the rules as data it makes it, theoretically, possible to modify them easily whilst retaining the same program for handling the rules.

Having built up a set of rules, it's reasonable to ask what PUFF does with them. If it had been our own expert system it would have gradually wandered through these rules drawing what conclusions it could as and when it could. PUFF (and MYCIN), however, don't work like that for they are much smoother.

In general, there are two methods of handling sets of rules - usually referred to as forward and backward chaining.

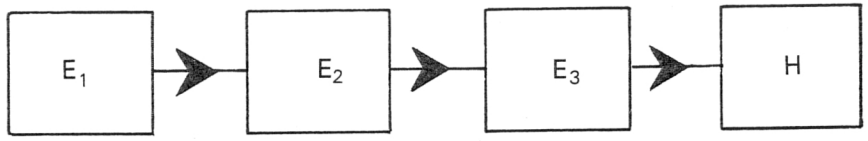


Fig. 9.1
Forward chaining

IF E_1,THEN: IF E_2 THEN: IF E_3 THEN:H
so all of E_1,E_2 and E_3 have to be established in order to establish H. H, as shown in Fig.9.1

In a forward - chaining, or data - driven, strategy this is exactly how it does proceed. The system is given E_1, after which it is given E_2, after which it is given E_3 - after which it can deduce H.

In a backward - chaining, or goal - driven, strategy the system first considers H and wishes to establish whether or not H is the case. Looking backwards it finds it needs to know E_3 to establish H. And to know E_3 it needs to know E_2. and to know E_2 it needs to know E_1. So it requests data on E_1.
After which it can proceed forwards again to H.

The difference is most apparent when there are a large number of different conclusions (goals, H) at which the system could arrive and a variety of routes to each goal.

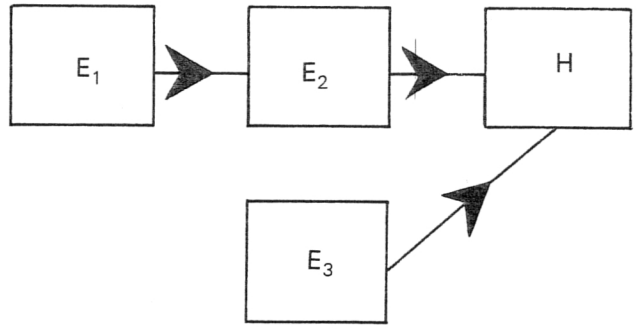


Fig. 9.2
Alternative reasoning

In Fig. 9.2, H is most readily established by asking for E_3 rather than by starting with E_1.

A in this case Forward Chaining would start with E_1 to give E_2 to give H. Backward Chaining would start with H and, then, look to E_2 or E_3. Seeing E_3 as the quicker route it would obtain E_3 to give H, ignoring E_1.

The system we described earlier used forward chaining inasmuch as it worked out what it could from what it had, and having done that it then went on to work out a bit more on the basis of what it had just done. Forward chaining, as its name implies, involves moving forward through the rules all the time driven by the features which are present in the data it's given.

Backward chaining, on the other hand, works backwards (surprise!) and is much more purposeful in its behaviour.

It starts off by putting up a hypothesis - Is the patient suffering from bronchitis? - and attempts to work out whether or not this is the case. To do this, it finds a rule, one of whose outcomes is bronchitis and checks to see what the antecedents of this consequent are (i.e. what would give the outcome 'bronchitis'). If it has data on these variables then it can come to a conclusion about the possibility of bronchitis. If it doesn't have data on these variables it chooses just one of them and tries to get data on it. Obviously, it can do this by either asking the operator to provide that item of data or it can do it by stepping back through the rules to find another rule which, if satisfied, would provide the data as one of its outcomes.

The method is essentially recursive and (in case you were thinking of it) it isn't particularly easy to code recursive techniques in BASIC.

But conceptually, if you think of our own system, it's a bit like going to the last node in the system (or some node which has an outcome labelled as a goal state) and then trying to get values for all of its inputs by stepping backwards through all of the preceding nodes that provide input to this final node.

The argument in favour of backward chaining is that it gives the system a sense of direction. It isn't just trying to find out anything it can in an *ad hoc* fashion - it's specifically trying to establish whether or not a number of important things are the case.

With a well-formulated set of rules it's highly arguable whether it matters much which method is used. But if you happen to have an expert system that has been collected over a period of time and which contains (shall we say) a wide assortment of rules of varying degrees of usefulness with a wide variety of outcomes of varying degrees of interest then it could well be an advantage to have a system which could get to the point as quickly as possible rather than establish the truth or otherwise of absolutely everything under the sun every time you used it.

As a final point on PUFF, one might wonder what benefits were gained by writing the program. Why not just leave a subject like this to human experts?

The obvious answer is that there's a bit of a shortage of experts in some fields. And, in many ways, this answer is too obvious.

After all, they could have written a book, or manual, on the subject and left it at that. Why bother with a computer program?

Well, in part there's some truth in this suggestion and there's probably also some truth in the suggestion that expert systems are being written simply because they're interesting to write.

But it's very interesting to note that early versions of PUFF, as with other expert systems, didn't work too well. Doubtless the human experts thought they knew what they were doing when they formulated the initial rule set - but, apparently, they didn't know all that well. So: if they'd written a book on the subject, what would it have been worth?

In a way, one could make this general-purpose point about computers as a whole - That they are really great Thought Machines. If you can think of something, some way of doing things, you can try to program it on a computer. If the program doesn't work too well then you can be sure that you have been unable to express what you wanted it to do - which implies that you didn't really understand the matter yourself. Or not as well as you might have done.

Only when the program is working can you really feel that you understand the problem fully - which makes the exercise of programming not just an exercise in educating the computer. It's also an exercise in educating oneself.

It is sometimes said that a person really has to understand a subject thoroughly to be able to teach it. By a similar token, one has to have a very good grasp of a subject to be able to teach a heap of chips the essence of that subject.

9.3 DENDRAL - chemical structures

Seeing as how expert systems are the latest thing it might come as a bit of a surprise to hear that DENDRAL dates back to 1965. There were, of course, computers in 1965 but it was the age of the transistor, paper tape and punched cards rather than the age of the chip and the screen.

All of which has nothing to do with DENDRAL itself - except to note that this must be the oldest, best-established, expert system in the world. Or, at least, the oldest system that's advertised as being an 'expert'.

Like PUFF, it comes from Stanford University and is a joint effort between the computer scientists and a bunch of human experts - this time in the field of chemistry. The basic idea is this:

When a chemist has prepared a substance he frequently wants to know what its chemical structure is and there are a number of ways he can find

out. First, he can make some intelligent guesses using his own expert knowledge. Next, he can put some of it in a spectrometer and study the resulting spectrum lines to refine his initial guesses. In many cases this will enable him to pin down the exact structure of the substance and everybody's happy. The problem is that this all takes time and a fair bit of expertise of the human variety. And this is where DENDRAL comes in - it's an attempt to automate the process of deducing the correct chemical structure.

In very broad outline the process is just the same as that used by our own system or PUFF. The user presents DENDRAL with some information about the substance plus the spectrometer data (infra red, nuclear magnetic resonance and mass spectrometry) and DENDRAL comes up with a 'diagnosis' in the sense of an appropriate chemical structure. But you only need to go marginally deeper into the workings of DENDRAL to find that the differences are so great as to almost mask the similarities altogether. The problem really arises because of two main facts. One, the structure of chemical substances can't easily be described in simple words. Two, the number of possible structures is enormous - literally millions of possibilities exist.

So, let's look at the first item - the description of the structures.

If you've got just a passing familiarity with chemistry you'll have come across pictures of chemical structures. Those graph-like drawings that show atoms and bonds. You might, for instance, have seen a picture of a benzene ring - if so, then you know roughly what it's like. If you only had a benzene ring and one or two other structures you could refer to them by name and proceed very much as if you were diagnosing the presence of, say, bronchitis given certain data. The problem is that there are so many different interconnection possibilities that it just isn't feasible to name them all. The only way to proceed is to draw them. And some of them are so complex that the drawings are very far from simple. The way around this problem is to describe each structure in terms of a graph with a variety of nodes and links. By doing this any structure that might exist can be described in one all-encompasing 'language'.

So far so good. But a point worth noting is: that, having done this, the same 'language' could not readily be used to describe, say, medical diagnosis. The language of graph theory simply wouldn't be appropriate here. It is occasionally said that the idea of an expert system is to provide a general-purpose reasoning program. This program could become expert in any area it chose simply by unplugging a set of rules on one subject and plugging in a new set of rules on another subject. DENDRAL is a very good example of the extent to which this ideal would be hard to realise because

it highlights the extent to which one descriptive language might suit one problem and not another. It would be a major inconvenience (if not a practical impossibility) to either force medical diagnosis into the language of graph theory or to force the description of chemical structures into the language of medicine. And to the extent that different subject areas are best suited to different descriptive languages we find that it is necessary to adopt a specialised approach for each subject area.

The second point concerns the number of possible structures. In medical diagnosis it's feasible to hold all of the possible outcomes in memory all of the time and the same is true of some other fields. But when the number of possible outcomes runs into millions this clearly is never possible. So the problem then becomes one of choosing the correct structure when the machine doesn't 'know' in detail just what structures are possible.

The way DENDRAL solves this problem is to generate possible chemical structures at run time and then test them to see if they're the right ones. If it did this with no constraints on its behaviour it could generate all possible chemical structures and would be back with the initial problem - simply, that there are too many of them. It gets around this problem by generating only a small subset of all possible chemical structures.

Effectively, you can think of DENDRAL as being in two parts. Almost, two separate expert systems in one.

The first part contains a set of rules for generating possible chemical structures. The input data to this part consists of a series of statements made by a chemist which provide some clues about what structures are likely in this instance. This is, in a way, very similar to the systems we've looked at so far.

The output from this first part, though, is not one single answer. It is, usually, a whole series of possible structures - the program is unable to say exactly which is the right one.

The second part of DENDRAL then takes each of these structures in turn and uses a second expert system to work out, for each, what spectrometer results it would give if this substance actually existed and was actually placed in a spectrometer. The input to this second expert is some program-generated chemical structure; the rules it uses are a series of rules derived from real chemists which express the behaviour of a spectrometer; and, the output is a simulated spectrometer response. At this point we recall that a further input to DENDRAL was the actual spectrometer readings which actually came from the substance under investigation - and DENDRAL compares its hypothesised simulation

results with the actual results to see if they're the same. If they are then this might be the right structure - if not, this structure can be discarded and another one generated.

The process is one of constant pruning to keep the number of possibilities under consideration as small as possible at any one time. Unlike some expert systems, DENDRAL isn't a 'toy'. It doesn't exist just to test a theory about expert systems - it actually has a real use in identifying chemical structures and over two dozen scientific papers have been written using the results of DENDRAL working on real problems. In its field it is said to rival human experts. As such, it certainly gives cause for optimism that other, useful, systems can be produced for use in other fields. But the extent to which they could make use of DENDRAL's methods is debatable.

Certainly, the overall scheme of narrowing the search area down with a few initial constraints, generating possible solutions, and testing these possible solutions against some other criteria - that might be applicable in other fields.

But the precise DENDRAL code in which solutions are described in terms of graph theory and tested by means of spectrometer simulation - that might be a bit difficult to transport to another field.

And this seems to be a fairly general point: That simple expert systems can be fairly easily converted to work in another field but are only of limited use in any field. And systems which have a very real use have this usefulness because of their complexity and are very domain-specific. By making them solve one particular set of problems really well they become highly specific to that particular problem area and become difficult, if not impossible, to adapt to other areas.

As a further illustration of this point, consider META-DENDRAL. You recall that DENDRAL has a system for simulating spectrometer results from a given chemical structure - well, META-DENDRAL is an expert system which was used to build that simulation. The problem was: How to simulate spectrometer results? What are the rules which an expert system should have so that it could do this? And that's what META-DENDRAL was built to find out. It works, roughly, the same as DENDRAL inasmuch as it is able to generate a whole series of possible rules which it then re-applies back on to the input data to see if each rule would explain the results.

Specifically, it receives as data some spectrometer output and a graph description of the substance that gave rise to that output. It then starts to

generate a series of rules which might be applied to that spectrometer data and, for each rule, applies it and checks its own output to see if it's the same as the given structure. If it isn't, it generates a different rule. If it is, then that rule is a possible rule.

By working over a large number of examples META-DENDRAL was able to generate a set of rules which would show the likely spectrometer output from a given structure.

In a (very rough) way this is similar to the system described in the early part of the book in which we just presented the expert system with a series of examples with known outcomes and let it work out a set of rules for itself. And it goes close to the heart of one of the main problems in the field - that when you start building an expert system you often don't really know what rules should be used at all and need some, preferably simple, way of finding out.

We noticed, with PUFF, that human experts' knowledge of the rules they use is often pretty shaky - and that's in a field in which there actually are human experts. But why should there be any human experts in the field of META-DENDRAL? What person would spend his time working out the likely spectrometer output from a substance of which he already knew the molecular structure? After all, if you have such a substance, you could always put it into a spectrometer and solve the problem that way. And, if you don't have such a substance, who cares what sort of output it would give? The need to have an expert in META-DENDRAL's field only arose because of the existence of DENDRAL itself and there wasn't really any expert to turn to for help - so everybody started from square one.

The problem is an interesting one. Most expert systems carry the knowledge of human experts in some form or another. But a system that can acquire its own knowledge in this way relies much less heavily on any pre-existing body of knowledge and is actually likely to increase our store of knowledge, not just about specific cases but about the entire domain in which it operates.

Consider the earlier example of weather forecasting. Maybe you know nothing about weather forecasting. Well, that doesn't matter too much. You can still set up a system and train it with a series of examples of actual weather and, in time, it's likely to get the idea and predict the weather to a reasonable extent. Now, the fact that it can do this means that it actually has a set of rules in there which are better than any rules you yourself have for forecasting the weather. And if you then poke around inside its rule set you would stand some chance of learning something about weather forecasting for yourself. A trivial example, possibly, but an interesting principle - and one which META-DENDRAL puts to good use.

9.4 PROSPECTOR - searching for minerals

Whereas it must be nice to be able to heal the sick, and doubtless could be nice to be able to deduce a molecular structure, there's not really much doubt that nicest of all would be to be able to discover gold in them thar hills. Real gold, that is. The sort you can spend.

Now, PROSPECTOR doesn't help in actual prospecting for gold for its originators have confined their activities to much duller though still valuable deposits. But the principle's the same. PROSPECTOR is an expert system designed to help in the hunt for commercially-exploitable mineral wealth. And, as such, it's interesting.

Traditionally - one supposes - the human expert in this field loads his mule with a few pots and pans, makes enough sandwiches to last him through the winter, and heads up into the hills to apply his expertise. Come Spring, having used his expert judgements, he is able to stagger into town, exhausted, and file his claim to a piece of goldbearing territory. Immediately after he does this he is, usually, shot leaving only his daughter to avenge him and get the gold back. The rest of the plot is pretty familiar (she does get his gold back, along with the hand of one who helped her, and lives happily ever after).

But the real question, which is never satisfactorily answered in most accounts, is: what did he actually do when he was up in the hills?

So begins the story of PROSPECTOR. For the first stage was - as with all expert systems - to find out what real human expert prospectors do when they're looking for gold (or some duller, but still valuable, deposit). And the answer, in general, is the usual series of inferences and deductions, some certain, some probabilistic, from which the expert forms a judgement on the matter.

Just in case you're wondering, the method doesn't consist of pointing the computer system at a vast tract of territory and asking it where one should look for gold - that's a bit vague for a computer. It consists of specifying an exact location about which certain facts are known and asking for an estimate of the probability of a certain deposit occurring at this location. It's a bit like medical diagnosis in some ways. In medical diagnosis you present the expert (human or machine) with a patient and ask what's wrong with that particular patient. You don't just ask the expert for a diagnosis of patients in general.

And, in some ways, the prospector can proceed as does the medic. Like MYCIN, for example, PROSPECTOR contains a large number of rules

concerning the various things which might be observed and the things which might be deduced from them. It then proceeds by backward chaining - hypothesising that a particular outcome might be the correct outcome and working backwards through the rules to see if it can justify this outcome.

Unlike medical diagnosis though, the result isn't a simple statement on the matter.

Consider. Given a patient suspected of being ill, what one wants to know is: what is he ill with? One of several outcomes has to be chosen. It's not very important really what the exact probability of the diagnosis being correct should be. It's simply a matter of getting that diagnosis which is the most likely.

Now contrast this with mineral prospecting. In any given situation any given mineral is likely to occur in some quantities - albeit minute - so the conclusion that a certain mineral exists isn't very helpful. What the user of a prospecting system wants to know is just how much of the mineral is present. Maybe the exact quantities aren't too important - it would be quite good enough to know that there was 'a lot'. But the probability of there being a lot there is very important. A simple yes/no answer won't do when the cost of digging up the terrain to test the system's opinion is high.

The subject of risk analysis has some bearing on the matter, inasmuch as there is a cost associated with a wrong decision.

Suppose you have a patient and you diagnose an illness in that patient. Now, the diagnosis might be right - in which case, by treating for that illness you've done the right thing. Good. And with some probability the diagnosis might be wrong - but, typically, it won't do the patient much harm if you still treat him for that illness. (Don't amputate limbs on the advice of a computer, though.)

The essence of medical diagnosis is that it's important to uncover, and treat for, every possibility and doesn't generally matter too much if the possibility doesn't turn out to exist in fact.

With mineral prospecting the situation is slightly different. Obviously, you'd like to dig in all the places where you might find gold just in case you happen to be lucky. But, in some ways, it's more important not to dig for gold in places where there isn't any.

Digging up the scenery takes time and costs money - and while you're digging up one bit, in general, you aren't digging up another bit.

The trick is to find the most likely place and dig there - and that's why an exact estimate of probabilities is important here in a way it wasn't in medical diagnosis. It's not that gold is more valuable than health (even though it is), it's just that the equations on which you take action are different.

Having said this, it's interesting to see how PROSPECTOR handles the matter of probabilities in coming to its conclusions - arguably, its methods are the best worked out of any current expert system.

The simplest cases occur with those rules that express logical relations. These are of the sort IF x THEN z in which z follows necessarily from x. Now that's easy, and it's still easy if you associate a probability with x because you can argue that if the probability of x is p then the probability of z is p also.

But, in general, for x to have a single argument would make that rule quite trivial. More usually we would replace x by a more complex term, say, (x AND y) or maybe (x OR y).

In the case of ANDed relations in which the individual elements have probabilities associated with each of them, PROSPECTOR takes the minimum of the values and assigns this minimum probability to the outcome. So, if the probability of x was 0.1 and the probability of y was 0.2 then the probability of z would be 0.1. It's easy to see why this method is chosen - for z to be true both x and y have to be true, which is a tight constraint so you take the minimum value.

On the other hand, for items joined with OR the maximum value is chosen because either x or y will cause z, which is a very loose constraint.

Actually, this method isn't entirely free from criticism - again, you could glance at the section on Probabilities. To give an idea of the error, if x and y are both independent variables then P(x and y) = P(x)P(y) i. e. the product of the two. If x and y aren't independent but are exactly correlated one with the other then P(x)=P(y)and P(x and y) = P(x) = P(y). And, in general, for partially correlated variables the truth lies somewhere in between.

Also, for the OR relationship independent variables would give P(x or y) = P(x) + P(y) - P(x)P(y). And, if they were completely correlated then P(x or y) = P(x) = P(y). With the actual value for partially correlated variables lying somewhere in between.

So, PROSPECTOR's method is a bit *ad hoc* but it's still a method which gives an answer in the appropriate range.

The next, and more interesting method, is one in which a series of assertions (rules) occur which each contribute something to the probability of some hypothesis.

For instance, we might say that if there's gold lying around in big lumps on the ground then there's gold in them thar hills with probability 0.9.

By itself this statement is much the same as ones we've come across earlier. The difference here is that we're really only concerned with the one hypothesis - that there's gold in them thar hills - and, therefore, a very great number of statements will occur which have some bearing on that hypothesis, either supporting it or contradicting it. The problem then becomes one of how to keep score of all of these probabilities.

The method PROSPECTOR uses is a neat application of Bayes' formula (see section 2.5) for assessing the prior and posterior probabilities of an event occurring - and 'event' here can be anything, such as the hypothesis concerning gold and hills being true.

In outline, each hypothesis starts off with an initial (prior) probability of being true, say P(H). So, the prior probability of there being gold in them thar hills might be, say, 0.1. P(H)=0.1.

Now, there you are in your prospector's hat, sitting at the screen presenting the system with an extra piece of evidence. Having got this extra piece of evidence ("just found a piece of gold the size of your fist", say) the probability P(H) changes to become P(H:E) - the probability of the hypothesis given this new evidence. So the system can update its old probability value with an assignment statement: P(H) = P(H:E) and then proceed to check out a new item of evidence.

The question then is: how to calculate P(H:E).

Well, the Rev. Bayes had an answer. It is:

P(H:E) = P(H)P(E:H) / (P(H)P(E:H) + P(not H)P(E: not H))

And, in some ways, the best thing to do is to put this into the program and forget it. If you can't persuade yourself to do that, then try:

P(H:E) = LS.P(H) / (P(not H) + LS.P(H)) which, at least, looks easier. And LS = P(E:H) / P(E: not H)

All of these formulae mean the same thing in fact and the explanation goes like this.

LS is the likelihood ratio (a well-known quantity, but only to statisticians). It is the ratio, in this instance, of the probability of receiving this bit of evidence given that the hypothesis is true divided by the probability of getting this same bit of evidence given that the hypothesis isn't true.

So - the hypothesis is that there's gold in them thar hills. Now, if this is true the probability of picking up a piece of gold the size of your fist is, maybe, 0.3. And, if it isn't true the probability of the same event falls dramatically to, say, 0.1. So the likelihood ratio has the value 3.0.

If we turn then back to the formula for P(H:E) we get

$$P(H:E) = 3\ P(H) / (1 - P(H) + 3\ P(H)) = 3\ P(H) / (1 + 2\ P(H))$$

And so, if there was a fifty:fifty chance of gold being in the hills beforehand P(H)=0.5 then P(H:E) = 1.5 / (1 + 1) = 0.75.

In other words, things are looking up.

This gives a new value of P(H) for this hypothesis and, as more evidence accumulates, new values of P(E:H) will continually modify P(H).

The story doesn't quite stop there, because there are always a few pessimists around who will note that some evidence points to the fact that the isn't any gold around. Specifically, you might find that there aren't any pieces of gold lying around the size of your fist and you want to be able to allow for this. The calculation is the same as before except that you consider not-E instead of E, to indicate that the evidence was lacking. And you then calculate the likelihood ratio and adjust P(H) as before. The main point to note is that you need a new set of probabilities - a new likelihood ratio because the old one was based on the presence of the evidence. So, if we take

$$LN = P(\text{not E:H}) / P(\text{not E: not H})$$

as the likelihood ratio associated with the lack of a certain piece of evidence we can calculate a new P(H)=P(H: not E) as before except that we use LN instead of LS.

Suppose, for instance, that you haven't found that bit of gold. Then the probability of not finding a lump lying around given that there is gold in the hills might be 0.9, say, and the probability of not finding gold lying around given that there's no gold in the hills might be 1.0. Giving LN = 0.9 / 1.0 = 0.9. So the absence of gold lying around would reduce the probability of there being gold in the hills - but not by very much. In the example above it would reduce P(H) from a value of 0.5 to a value of 0.47 approximately.

Of course, if the system asked you about lumps of gold lying around and you said that, yes, there were, then there'd be no need for it to then ask you if it was true that there weren't any lumps of gold lying around. Effectively, the reason for having the two values LS and LN is simply to put together the results of the two questions - the positive side and the negative side, without having to be so boring as to ask each question separately.

In general, the rules in PROSPECTOR are all in this form IF...THEN (LS,LN) so that each rule is set up with a likelihood ratio both for a positive response and a negative response. These ratios are calculated as just described and originate in the minds of expert, human, prospectors. The designers of the system having asked the experts questions like: If there was gold in them thar hills what do you reckon the chances would be of it lying around in big lumps? And: If there wasn't gold in them thar hills what do you reckon the chances would be of there not being any lying around?

Four questions in all to cover the full range of possibilities, to give values for:

P(E:H)
P(E: not H)
P(not E:H)
P(not E: not H)

Obviously, in order to keep things simple, we've considered a rather special case - that in which the user of the system actually knows the answer to the question the system asks.

After all, any idiot ought to know if he's got a lump of gold in his fist or not.

But, in general, in real life, the answers are much less certain.

Prospectors of other minerals tend to ask pretty specific questions, like: Has hornblende been pervasively altered to biotite?

And, really, one sympathises with anyone who isn't altogether sure of the answer to that one.

The PROSPECTOR solution is to give the user a scale from -5 to +5 in which to answer. A reply of +5 is definite yes, and -5 is definite no.

Typically, the user will answer somewhere in between and PROSPECTOR takes account of this by readjusting P(H) with a little bit of LS and a little bit of LN by a system of linear interpolation. You can think of it as a linear scale with LN on the left and LS on the right. Accordingly, as the user's response varies between -5 and +5 then P(H) is adjusted with the value, L say, which is picked off from this sliding scale between the two.

As it stands, the system that PROSPECTOR uses is pretty elegant in theory and, according to reports, pretty good in practice. But, like most experts, it isn't quite so modular in design as it at first appears.

The main difference occurs when the user starts to respond with answers that aren't simply yes or no. For instance, the system might want to know the estimated age of a lump of rock and the answer would be a figure in years. This leads to a situation in which the values for LN and LS aren't simple values. In general, likelihood ratios are functions of a variable and these functions have to be stored somewhere and calculated at the time the input is given. It's not a big problem - but it's a departure from strict modularity. Also, the strictly logical questions require a slightly different form of code to the more general probabilistic items.

The result is that, although the principles used in PROSPECTOR could be adapted to other areas, the exact code might not be easily modifiable - a comment that could be made about many expert systems.

And, to a very large extent, the bulk of the work on building the system, seems to consist of obtaining the rules from the human experts on the subject - again, a comment that could be made of most other systems.

But the overall impression (again, with most systems) is that once the human experts have been interrogated and a set of rules drawn up the hardest part of the job is over. For, after this has been done, the form in which the system should be programmed becomes, to some extent, apparent. Or, at the very least, some possible ways of proceeding become very unattractive and some other ways tend to stand out.

It's all a bit like the distinction between systems analysis and programming. If by 'programming' you mean doing the entire job then it can be quite complicated. But if programming simply entails coding up a structure predetermined by systems analysis then it's fairly straightforward. In the field of expert systems the big problem lies in uncovering this overall structure prior to the actual coding for each separate area of expertise.

9.5 Some other examples

The problem with giving examples of expert systems is that the very definition of expert system is sufficiently loose to enable one to add in just about anything. This doesn't mean that the definition is completely useless - it just means that it's flexible. You can stretch it a bit (in which case the list of current expert systems become nearly endless) or hold it rigid (in which case there are hardly any expert systems in existence at all).

And it's not just us who have this problem. At the time of writing the UK's Central Computer and Telecommunications Agency (CCTA) had got sufficiently jumpy about the prospect of Great Britain being swamped by cheap Japanese expert systems that it commissioned a survey to find out who was doing what in the field. This survey (and no disrespect is intended to any involved parties) consisted of a questionnaire which began with a definition of expert systems (the one given at the front of this book) and then, more or less, asked respondents whether they had one and, if so, what it did. It's a sensible approach in many ways but it would have been more sensibly applied to a subject about whose existence there was less doubt.

For instance, towards the end of this book you'll find a Bayesian inferencing program which can be readily used for medical diagnosis. So: put it on your micro and you then have an expert system and can tell the CCTA about it. If everybody did that UK usage of expert systems would, apparently, rocket. Or, maybe, do the same with the system in the first part of this book. Or, maybe, sinking even lower, just write a program to do something (anything) and decide that that's an expert system. And who's going to argue with you?

The point is that asking someone if they have an expert system isn't the same thing as asking them if, say, they've got a colour TV.

Still, be that as it may, the following is a list of some systems which are claimed to have a large knowledge-based component.

System Name	Purpose
MYCIN	Medical Diagnosis
PUFF	"
PIP	"
CASNET	"
INTERNIST	"
SACON	Engineering Diagnostics
PROSPECTOR	Geology Diagnostics
DENDRAL	Chemistry
SECHS	Chemistry
SYNCHEM	Chemistry
EL	Circuit Analysis
MOLGEN	Genetics
MECHO	Mechanics
PECOS	Programming
R1	Configuring Computers
SU/X	Machine Acoustics

VM	Medical Measurements
SOPHIE	Electronics Tuition
GUIDON	Medical Tuition
TEIRESIAS	Knowledge Acquisition
EMYCIN	"
EXPERT	"
KAS	"
ROSIE	Building Expert Systems
AGE	"
HEARSAY III	"
AL/X	"
SAGE	"
Micro-Expert	"

There are a number of interesting points to note in this list. And, doubtless, the first point to note is the number of expert systems which are involved in building other expert systems - and maybe this helps to show the difficulty of definition. Consider, for a moment, compilers. These are computer programs and their function is to help people write computer programs. And, by allowing a definition of expert systems that includes expert systems that help people write expert systems, aren't we, somehow, getting a definition that's really about as loose as the term 'computer program'? Maybe it doesn't matter if we are. After all, we could just sit back and say: "They're useful, who cares what they're called?" And that attitude may well, in the end, prove to be the best one to adopt - otherwise there's a danger of being so pedantic that we finish up criticising other people's work simply because it doesn't fit some preconceived label.

The next interesting group of systems are those used in Knowledge Acquisition. We've already noted that getting the expert knowledge into the system is one of the hardest tasks and one for which computer help would be handy - so here's some computer help.

Not, of course, that we haven't got our own help; our learn-by-example system is a Knowledge Acquisition System in its very own right.

If you've got your own expert system (the one described so far) running you might be interested to see if it can run in any of the problem areas listed above.

Medical diagnosis should be fairly straightforward. Use symptoms as the input variables and illnesses as the outcomes and throw some examples at the system to see how it gets on (though you might find the later, Bayesian, system rather more effective for this).

Engineering diagnostics - yes. You could try building up a system to show you why your car won't start.

Geology - maybe. It depends rather on your knowledge of geology. The advantage of medicine is that you can buy medical encyclopaedias fairly cheaply which give information on lots of illnesses to work with. There aren't very many home textbooks on mineral prospecting, unfortunately.

Anyone with an interest in palaeontology could try classifying fossils using our expert. For instance, a lamellibranch has an asymmetrical shell whereas a brachiopod has a symmetrical shell. So: it is a lamellibranch or a brachiopod? The snag is that, having said that, you hardly need a computer to work out the answer once you've looked at the shell! More complex examples may well spring to the mind of the enthusiast.

Chemistry - yes. Chemical analysis (at school level) does involve a series of tests with specific results. So it could assist in some chemical analyses.

Circuit analysis - again yes. Because the system only deals with Yes/No responses it would be no good with analogue circuits (in which currents vary continuously) but it could very easily represent a digital process. Use a multi-node system and let each node represent one component whose inputs are either 0 or 1. Interconnect the various nodes and you could have an expert system to represent a whole boardful of chips. If certain conditions were good (i.e. were consistent with the designers' intentions) then these could be monitored by another node designed to judge good conditions from bad ones. So the expert could simulate the 'chip' operations at the same time as monitoring the operation for error conditions.

Many of the applications listed above will tend, however, to leave our own expert system somewhat standing - largely because ours is too general-purpose to be able to be tailored to some of these specific applications.

Take MECHO, for instance. This is an expert system which can give intelligent answers to complicated mechanical problems. Suppose, for instance, that you have a system of pulleys, strings and weights. In place of one weight there is an empty pan hanging by a string. With the pan empty the system is not in equilibrium and the question is: what weight should be placed in the pan to bring it into equilibrium? The problem is fairly familiar in school physics. To answer the question the system needs to understand the arrangement of pulleys, the relevant laws of physics, and be able to use this knowledge to get an answer. MECHO can do this. And, in a very, very crude way our multi-node system might also do it. Using each node to represent a pulley the inputs could be the weight

hanging from that pulley and the outputs (outcomes) could be the strings supporting that pulley. And the problem is that our system with its Yes/No outcomes could only (once it had got past the first pulley) say whether a given string had a force acting on it or not - not the exact force that was acting on it. Which makes an exact solution impossible.

You might try to get around it by 'using' standard weights - like, for instance, presence or absence of a weight altogether. And then add a 'monitoring' node to check for equilibrium at some point(s) as its input and then outputting additional weights here and there according to its inputs until the system balanced. But it might (might!) lack a certain precision.

On the other hand, of course, it might work. There's really no substitute for putting together a system that you think might do something and then watching to see what it actually does. After all, that's what everyone else does! That's what is known as Research.

CHAPTER 10

10. A Rule-Based BASIC Expert

10.1 A system that works backwards

Although the system described in the early parts of this book may be fine for you, the previous chapter shows that there are some alternative methods that might be more useful in certain circumstances. In this chapter there is a practical alternative for you.

In a sense, one of the main problems in building an expert system seems to be that even if all of the necessary knowledge is readily available it's in a form which is the wrong way round.

Suppose you want to build a system which is expert in the field of medical diagnosis.

Now: you hardly need to build a system based on learning by example, as we did in the case of weather forecasting, because there's lots of information readily available which should allow you to go to a much more direct solution. The problem really is that the information appears to be the wrong way round.

Get a medical encyclopaedia and look up the entry for influenza, say. You'll find that all of the symptoms are given, and that there isn't any argument about these symptoms. In other words, given the symptoms, an accurate diagnosis could be made every time.

But, to use information organised in this way rather suggests that what you should do is to pick up a patient, decide he has influenza, then look in the encyclopaedia to see if he has the right symptoms and, somehow, this seems all wrong.

What you want to do is to pick up a patient, decide what his symptoms are, and then look these symptoms up to see what he's suffering from - and the encyclopaedia doesn't seem quite the right way of doing this. Instead of one illness with lots of symptoms we want a system which shows a group of symptoms followed by one illness. And that's what we'll put together now. The ideal being a situation in which you can, in some particular field, simply throw a lot of definitions at the machine in such a way that it can use these definitions rather like a human expert might use them.

This is what, of course, programs like PUFF, DENDRAL and PROSPECTOR try to do. So there's no harm in us giving it a try.

We'll use a Bayesian inferencing system to allow for the fact that most information isn't absolutely certain, but probabilistic, and to allow for the fact that it sounds pretty good to say you're using a Bayesian inferencing system when you're telling people about it all. And we'll place the main emphasis on the form of the information you're going to give the program about the field in which it's supposed to become expert because collecting this information is likely to be he hardest part of the job.

So, to get going, we'll start coding.

```
2000 DATA SYMPTOMS
2010 DATA 1, SYMPTOM 1
2020 DATA 2, SYMPTOM 2, END
```

This is the form in which we'll keep the symptoms. By saying 'symptoms' it sounds as if we're exclusively concerned with medical work - but it might be anything really. The essence of the matter is that there's a lot of questions the computer might ask and that these questions are held as strings SYMPTOM 1, SYMPTOM 2, etc.

For instance, SYMPTOM 1 might be the string: DO YOU COUGH A LOT? Or, if you were trying to fix a wayward car: ARE THE LIGHTS DIM?

Organised in this way, you can type in a lot of questions very fast.

Now we have the illnesses.

```
1000 DATA ILLNESSES
1010 DATA ILLNESSES 1,p, (j,py,pn,) 999
1020 DATA ILLNESSES 2,p, (j,py,pn,) 999
```

This is the form in which we'll keep the illnesses. They needn't actually be illnesses though. They can be any outcomes and each DATA statement contains one outcome and all the information relating to it.

To go through the items one at a time, the first is the name of the outcome - say INFLUENZA. The next item, p, is the prior probability of that outcome P(H) - this is the probability of this outcome occurring given no further information at all. We then have a series of repeated items with three elements. The first element, j, is the number of the relevant symptom (or variable if you want to call it something else). The next two items are P(E:H) and P(E:not H) - the probabilities of getting a Yes answer to this variable given the outcome is true and the probabilities of getting a Yes answer if it isn't true. The last item is a stop code - it's there so that the program can tell when it's come to the end of the details on one particular illness.

For example:

```
1010 DATA INFLUENZA, .01, 1,.9,.01,2,1,.01,3,0,.01,999
```

This says that, in the case of influenza, there is a prior probabilty P(H)=.01 of any random person having this illness.

Now suppose that the program asks question 1 (symptom 1). We have P(E:H)=.9 and P(E:not H)=.01 which says that if the patient has influenza then nine times out of ten he'll answer Yes to this question and, if he doesn't have influenza, then he'll only answer Yes in one case in a hundred. Obviously, a Yes answer supports the hypothesis that he has influenza. A No answer tends to suggest that he hasn't.

Similarly for the second symptom/probability group (2, 1, .01). In this case P(E:H)=1 which says that, if has has influenza, then he must have this symptom. He might have the symptom without influenza (P(E:not H) =.01) but it's not very likely.

Question 3 rules out influenza if he gives a Yes answer because P(E:H)=0. This could have been a question like - Have you had the symptoms for most of your life? Or some such.

It takes a bit of thought and, if you want good results, a bit of research to come up with reasonable figures for these probabilities. And, to be honest, getting this information in the first place is probably the hardest task - and one in which a computer can't help you much. But if you can get the information in this form then you can write a fairly general-purpose program to sort it all out.

Fundamental to this program is Bayes' Theorem, which states that:

$$P(H:E) = \frac{P(E:H)P(H)}{P(E:H)P(H) + P(E:\text{not } H)P(\text{not } H)}$$

The gist being that the probability of some hypothesis given a certain piece of evidence can be calculated from the prior probability of that hypothesis with no knowledge of the evidence and the probability of the evidence arising given that either the hypothesis is true or that it isn't true.

So, looking at our illnesses, we can calculate:

$$P(H:E) = \frac{py.p}{py.p + pn.(1-p)}$$

The process is that we start off with P(H)=p for each illness. The program asks a question and calculates P(H:E) depending on the answer. The answer Yes gives the above calculation. The answer No gives the same calculation but with (1-py) instead of py and (1-pn) instead of pn.

Having done this, this question is then 'forgotten' except inasmuch as P(H), the prior probability, is replaced by P(H:E). And the process carries on like that, continually updating P(H) as new information arrives.

Broadly, we can divide the program into a number of parts:

PART ONE
The program searches through the DATA statements to find out how many illnesses there are and how many symptoms. You could have told it this in advance, but it saves you having to count them all up if it can do it for itself.

At this point any arrays that are needed can be DIMensioned.

PART TWO
The program checks the DATA statements to find all the prior probabilities P(H). It also works out some rule values RV(I). The idea of these is to see which questions (symptoms) are the most important so that it knows which to ask first. If you calculate:

RV(I) = RV(I) +ABS(P(H:E) - P(H:not E))

for each question you'll get RV(I) showing values which represent the amount of change they can make to the probabilities of all the illnesses to which they apply.

PART THREE
The program finds the most important question and asks it. There are a number of ways you can handle the answer - you could just take Yes or No. You could also consider a Don't Know (which produces no change). More

complex - you could have a scale from -5 to +5 to express degrees of certainty in the answer.

PART FOUR
The prior probabilities are updated with the new values, given the new evidence.

PART FIVE
New rule values are calculated. Also calculated are minimum and maximum values for each illness based on the current prior probabilities and the supposition that either all the remaining evidence goes in favour of the hypothesis or goes against it. The idea is to see whether any given hypothesis might still be in the running or not. Those which aren't can be discarded. Those whose minimum values are above a certain level can be announced as possible conclusions.

The program then goes around to Part Three and continues until there's nothing more for it to do.

10.2 The BASIC program

Here is the BASIC equivalent of the above steps. For clarity, the Applesoft version is explained fully and the Spectrum version is listed.

```
 10  REM :FIND NUMBER OF ILLNESSES AND SYMPTOMS
 20  READ A$,A$
 30  IL = 0:RL = 0
 40  READ P,J
 50  READ PY,PN,S: IF S < > 999 THEN : GOTO 50
 60  IL = IL + 1
 70  READ A$: IF A$ = "SYMPTOMS" THEN : GOTO 90
 80  GOTO 40
 90  READ A$,B$: IF A$ = "END" THEN : GOTO 120
100  RL = RL + 1
110  GOTO 90
120  DIM P(IL),RV(RL),EO%(IL),MI(IL),MA(IL),IL%(IL),ER%(RL)
130  FOR J = 1 TO RL
140  ER%(J) = 1
150  NEXT
```

The code scans the list of illnesses looking for, and counting, the number of stop codes to give IL, the number of illnesses defined. It then reads the list of symptoms, counting them, until it comes to the word END in the DATA, giving RL as the number of symptoms listed.

The arrays DIM'd are:

P(IL) - which is used to keep a record of the current probabilities.
RV(RL) - which is used to keep a record of the 'value' of each symptom in terms of the amount of change it can induce in the probabilities of the illnesses.
EO%(IL) - a list of the number of relevant symptoms outstanding for each illness, this is reduced every time a symptom is queried.
MI(IL) - the minimum possible value which each illness can achieve.
MA(IL) - the maximum possible value which each illness can achieve.
IL%(IL) - a record of the number of symptoms listed for each illness.
ER%(RL) - a 'switch' which is set to 1 initially. After each query it is set to 0 to stop the same question being asked twice.

```
160  REM :FIND THE PRIOR PROBABILITIES AND RULE VALUES
170  RESTORE : READ A$
180  FOR I = 1 TO IL
190  READ A$,P(I),J
200  P = P(I)
210  READ PY,PN,S
220  EO%(I) = EO%(I) + 1
230  RV(J) = RV(J) + ABS (P * PY / (P * PY + (1 - P) * PN) - P * (1 - PY) / (P * (1 - PY) + (1 - P) * (1 -
     PN)))
240  IF S < > 999 THEN :J = S: GOTO 210
250  IL%(I) = EO%(I)
260  NEXT
270  REM :FIND THE MAXIMUM SYMPTOM AND QUERY
280  R = 0:HR = 0
290  FOR J = 1 TO RL
300  IF RV(J) > R THEN :HR = J:R = RV(J)
310  RV(J) = 0
320  NEXT
330  IF HR = 0 THEN : PRINT "NO FURTHER SYMPTOMS": END
340  READ A$
350  FOR J = 1 TO HR
360  READ P,A$
370  NEXT
380  HOME : PRINT ,"E X P E R T": PRINT ,"----------": PRINT : PRINT : PRINT "QUESTION :-":
     PRINT A$
390  ER%(HR) = 0
```

At this point the program has found a question and printed it on the screen. What you have to do now is to answer it.

In this program your response is held in RE and is on a scale from -5 to +5.


```
400  INPUT "REPLY ON A SCALE -5 (NO) TO +5 (YES)";RE
410  REM :UPDATE THE PRIOR PROBABILITIES USING THE RESPONSE
420  RESTORE : READ A$
430  FOR I = 1 TO IL
440  READ A$,P
450  FOR K = 1 TO IL%(I)
460  READ J,PY,PN
470  IF J < > HR OR EO%(I) = 0 THEN : GOTO 540
480  EO%(I) = EO%(I) - 1
490  P = P(I)
500  PE = P * PY + (1 - P) * PN
510  IF RE > 0 THEN :P(I) = P * (1 + (PY / PE - 1) * RE / 5)
520  IF RE < = 0 THEN :P(I) = P * (1 + (PY - (1 - PY) * PE / (1 - PE)) * RE / 5)
530  IF P(I) = INT (P(I)) THEN :EO%(I) = 0
540  NEXT
550  READ ST
560  NEXT
```

You might be a bit puzzled by the calculations for P(I) which vary depending on whether or not RE is positive or negative.

The idea is this:

If RE were +5 then the answer would be Yes and we calculate

$$P(H:E) = \frac{P(E:H)P(H)}{P(E:H)P(H) + P(E:\text{not } H)P(\text{not } H)}$$

and, if it were -5, we calculate

$$P(H:\text{not } E) = \frac{P(\text{not } E:H)P(H)}{P(\text{not } E:H)P(H) + P(\text{not } E:\text{not } H)P(\text{not } H)}$$

and, if RE were 0 we calculate

$P(H:E) = P(H)$.

And if it's somewhere in between the extremes we have a bit of one and a bit of another - the code given calculates just how much of one bit and how much of another with a bit of simplification and re-arranging thrown in to speed things up a bit.

If P(I) = INT(P(I)) then we can knock EO%(I) down to zero because we have a certain event - no more questions are needed on this item.

```
570   REM :FIND NEW RULE VALUES AND MINIMUM AND MAXIMUM PROBABILITIES
580   RESTORE
590   READ A$
600   MM = 0:MH = 0
610   FOR I = 1 TO IL
620   P = P(I)
630   A1 = 1:A2 = 1:A3 = 1:A4 = 1
640   READ A$,PZ
650   FOR K = 1 TO IL%(I)
660   READ J,PY,PN
670   IF ER%(J) * EO%(I) = 0 THEN : GOTO 740
680   IF PN > PY THEN :PY = 1 - PY:PN = 1 - PN
690   RV(J) = RV(J) + P * PY / (P * PY + (1 - P) * PN) - P * (1 - PY) / (P * (1 - PY) + (1 - P) * (1 - PN))
700   A1 = A1 * PY
710   A2 = A2 * PN
720   A3 = A3 * (1 - PY)
730   A4 = A4 * (1 - PN)
740   NEXT
750   MA(I) = P * A1 / (P * A1 + (1 - P) * A2)
760   MI(I) = P * A3 / (P * A3 + (1 - P) * A4)
770   IF MA(I) < PZ THEN :E0%(I) = 0
780   IF MI(I) > MM THEN :MH = I:MM =MI(I)
790   READ A$
800   NEXT
```

Of this, probably the calculation of the new rule values looks easy, but the bit about MI(I) and MA(I), probably isn't so clear.

First, notice that we tested PY against PN to see which was the biggest. If PY is the biggest - fine, that means that a Yes answer increases P(H) and a No answer decreases it. If it's the other way around we want the opposite of these answers and that means, when you're working with probabilities, taking the complement, i.e. 1-PY and 1-PN. This way we can phrase our questions any way around we like and, as long as we got the probabilities right, the program knows whether it has supporting evidence for a hypothesis or not.

What we do now is to consider MA(I) - the maximum probability possible and we calculate this by assuming that all the unanswered questions will eventually be answered in such a way as to support the hypothesis.

We still calculate P(H:E) in exactly the same way as we would have done if we'd been adjusting the probabilities. But, for the maximum possible value we regard E as not just the answer to one question but the answer to all the outstanding questions. So P(E:H), for instance, is the probability

that all the supporting evidence occurs given the hypothesis is true. And, assuming that each question is independent of every other question, this is the product of all the outstanding values of P(E:H).

P (all supporting evidence occurs: H) = $P(E_1:H)P(E_2:H)P(E_3:H)...P(E_n:H)$

which is the value we stored in A1.

So we have:

A1 = P(all supporting E:H)
A2 = P(all supporting E: not H)
A3 = P(no supporting E:H)
A4 = P(no supporting E:not H)

And, frankly, anyone that starts to get a little mixed up at this point isn't alone. The procedure I usually adopt is to write down what I think it means then go and mow the lawn. Then I come back and write down what I should have written down if I'd been thinking straight in the first place. Then I go and mow the lawn again. Then I realise that I hadn't seen it quite straight that time and alter the equations a bit. Then I go and mow the lawn again. And so on.

At the end of the day, the lawn looks magnificent and I have a few drinks at which point I realise I've got it wrong and, just before passing out, see what I should have done and forget to write it down.

The next day the process resumes.

At the end of the week I am nearly down to bedrock where once there was a fine lawn and am trembling with what might well be fatigue.

I then realise that the very first set of equations I wrote down were right but that I've lost the piece of paper I wrote them on.

You, too, can spend your life this way. It has the great benefit of qualifying you for a disability pension around 35 or less. Or, you can plan the whole thing systematically in the first place.

Anyway, having got A1, A2, A3, A4 you can calculate the minimum and maximum possible values as shown. All you need do is copy out the code and reflect on the fact that this treatise is built on human suffering beyond average comprehension.

Finally, we deduce the most likely conclusion, like this:

```
810   REM :SEARCH FOR A CLEAR WINNER
820   FOR I = 1 TO IL
830   IF MI(MH) < = MA(I) AND I < >  MH THEN :MM = 0
840   NEXT
850   IF MM = 0 THEN : GOTO 270
860   RESTORE : READ A$
870   FOR I = 1 TO MH
880   READ I$,A$
890   FOR K = 1 TO IL%(I)
900   READ J,PY,PN
910   NEXT
920   READ A$
930   NEXT
940   HOME : PRINT ,"E X P E R T": PRINT ,"----------": PRINT : PRINT : PRINT "THE MOST LIKELY
      OUTCOME IS ";I$
950   PRINT "WITH A PROBABILITY ";P(MH)
960   END
1000  DATA  ILLNESSES
1010  DATA  ILLNESS 1,.01,1,.8,.1,2,.2,.9,999
1020  DATA  ILLNESS 2,.5,1,1,.5,2,0,.1,999
2000  DATA  SYMPTOMS
2010  DATA  1,SYMPTOM 1
2020  DATA   2,SYMPTOM 2,END,END
```

What this does is to take the maximum of the minimum values possible and see if it's greater than the maximum of any of the other values.

If it is, you stop, having finished.

If it isn't this is because some other outcome could, just possibly, be more probable than this particular outcome and, as this is the most probable outcome, you can't draw any firm conclusion yet and you have to go back and query another item.

If this sounds a bit too much like hard work you could simply pick that outcome with the greatest P(I) on the basis of the questions asked so far and, if it's a high enough probability to satisfy you, decide that this is the correct conclusion. Or, correct enough, anyway.

Here is the complete program written in Spectrum BASIC:

Sinclair Spectrum listing

```
    2 REM Notice that the DATA st
atements and corresponding tests
 for stop codes are changed slig
htly for the SPECTRUM
    5 DIM a$(200): DIM b$(20): DIM
 i$(20)
   10 REM Find number of illnesse
s and symptoms
   20 READ a$,a$
   30 LET il=0: LET rl=0
   40 READ p,j
   50 READ py,pn,s: IF s<>999 THE
N GO TO 50
   60 LET il=il+1
   70 READ a$: IF a$( TO 8)="symp
toms" THEN GO TO 90
   80 GO TO 40
   90 READ p,a$: IF a$( TO 3)="en
d" THEN GO TO 120
  100 LET rl=rl+1
  110 GO TO 90
  120 DIM p(il): DIM v(rl): DIM o
(il): DIM i(il): DIM a(il): DIM
l(il): DIM r(rl)
  130 FOR j=1 TO rl
  140 LET r(j)=1
  150 NEXT j
  160 REM find prior probabilities
 and rule values
  170 RESTORE : READ a$
  180 FOR i=1 TO il
  190 READ a$,p(i),j
  200 LET p=p(i)
  210 READ py,pn,s
  220 LET o(i)=o(i)+1
  230 LET v(j)=v(j)+ABS (p*py/(p*
py+(1-p)*pn)-p*(1-py)/(p*(1-py)+
(1-p)*(1-pn)))
  240 IF s<>999 THEN LET j=s: GO
TO 210
  250 LET l(i)=o(i)
  260 NEXT i
  270 REM find max symptom and qu
ery
  280 LET r=0: LET hr=0
  290 FOR j=1 TO rl
  300 IF v(j)>r THEN LET hr=j: LE
T r=v(j)
  310 LET v(j)=0
  320 NEXT j
  330 IF hr=0 THEN PRINT "No furt
her symptoms": STOP
  340 READ a$
  350 FOR j=1 TO hr
  360 READ p,a$
  370 NEXT j
  380 CLS : PRINT AT 12,0;;"EXPER
T";"_______";"Question:";a$)
```

```
 390 LET r(hr)=0
 400 INPUT "Reply on -5 to +5 Sc
ale:";re
 410 REM update prior probabilit
ies
 420 RESTORE : READ a$
 430 FOR i=1 TO il
 440 READ a$,p
 450 FOR k=1 TO l(i)
 460 READ j,py,pn
 470 IF j<>hr OR o(i)=0 THEN GO
TO 540
 480 LET o(i)=o(i)-1
 490 LET p=p(i)
 500 LET pe=p*py+(1-p)*pn
 510 IF re>0 THEN LET p(i)=p*(1+
(py/pe-1)*re/5)
 520 IF re<=0 THEN LET p(i)=p*(1
+(py-(1-py)*pe/(1-pe))*re/5)
 530 IF p(i)=INT (p(i)) THEN LET
 o(i)=0
 540 NEXT k
 550 READ st
 560 NEXT i
 570 REM find new rule values, m
inima and maxima
 580 RESTORE
 590 READ a$
 600 LET mm=0: LET mh=0
 610 FOR i=1 TO il
 620 LET p=p(i)
 630 LET a1=1: LET a2=1: LET a3=
1: LET a4=1
 640 READ a$,pz
 650 FOR k=1 TO l(i)
 660 READ j,py,pn
 670 IF r(j)*o(i)=0 THEN GO TO 7
40
 680 IF pn>py THEN LET py=1-py:
LET pn=1-pn
 690 LET v(j)=v(j)+p*py/(p*py+(1
-p)*pn)-p*(1-py)/(p*(1-py)+(1-p)
*(1-pn))
 700 LET a1=a1*py
 710 LET a2=a2*pn
 720 LET a3=a3*(1-py)
 730 LET a4=a4*(1-pn)
 740 NEXT k
 750 LET a(i)=p*a1/(p*a1+(1-p)*a
2)
 760 LET i(i)=p*a3/(p*a3+(1-p)*a
4)
 770 IF a(i)<pz THEN LET o(i)=0
 780 IF i(i)>mm THEN LET mh=i: L
ET mm=i(i)
 790 READ s
 800 NEXT i
 810 REM search for a clear winn
er
 820 FOR i=1 TO il
```



```
 830 IF i(mh)<=a(i) AND i<>mh TH
EN LET mm=0
 840 NEXT i
 850 IF mm=0 THEN GO TO 270
 860 RESTORE : READ a$
 870 FOR i=1 TO mh
 880 READ i$,p
 890 FOR k=1 TO l(i)
 900 READ j,py,pn
 910 NEXT k
 920 READ s
 930 NEXT i
 940 CLS : PRINT AT 12,0;,"EXPER
T";,"_____";''"The most likely o
utcome is ";i$;''"with a probab
ility ";p(mh)
 960 STOP
1000 DATA "illnesses"
1010 DATA "illness 1",.01,1,.8,.
1,2,.2,.9,999
1020 DATA "illness 2",.5,1,1,.5,
2,0,.1,999
2000 DATA "symptoms"
2010 DATA 1,"symptom 1"
2020 DATA 2,"symptom 2",0,"end"
```

Although this program works reasonably well (a miracle of modern technology, really) a great deal of its performance depends on the list of questions you supply it with.

On a theoretical level, the calculations all proceed on the assumption that each question is independent of every other question - and, if they aren't, the performance will deteriorate.

For instance, if you ask "Do you have a temperature?" and "Are you feverish?" it's obvious that the two questions are very highly correlated. There simply isn't any point in asking both questions but, if you do, it will have the effect of upsetting the probabilities you'd associated with these items.

Also, you have to provide enough questions to be able to actually make some sort of diagnosis. To actually enable the system to come to a conclusion. That sounds obvious, but suppose you gave details of the Common Cold and Influenza and reckoned that both made you feel unwell, with a runny nose and a cough. Maybe the probabilties would be a bit different in each case - but they have to be sufficiently different to enable the program to actually see the difference. The trick is to go for clear,

unambiguous quesions which don't overlap each other and are capable of splitting up the problem as cleanly as possible.

Typically, you'll find yourself working like any other person who's developed an expert system.

You'll write the program and throw a handful of likely-looking rules at it. (The rules are those definitions you gave in the DATA statements.)

You'll then pretend that you have pneumonia (if it's a medical system; maybe, a flat battery if you've been producing an expert mechanic) and answer the questions the system gives you with this in mind.

The system will then produce a wrong answer or ask some stupid questions - so you start fiddling around with the questions and the probabilities until the performance improves. And, really, this process has nothing much to do with computers at all. It's all to do with understanding the subject in which you want the computer to become expert - and it can be quite interesting in its own right.

And, if it isn't you can always revert to the system described in the earlier part of this book, just sling a few examples at it and let it work out for itself.

10.3 A medical knowledge base

If you set up the Bayesian Inferencing scheme just described then there will be one thing lacking before it will do anything useful - a Knowledge Base. Now, strictly speaking, that's your job to provide it with some domain-specific knowledge. But doing so can be pretty time-consuming. So, just to help things along a bit, here is a Knowledge Base for the Domain of Medical Diagnosis. If you load this code into line 1000 and forwards and line 2000 and forwards it will provide the expert system with knowledge on nearly 100 different disease types and their diagnoses. The figures given are, roughly accurate and although it won't enable you to practice medicine it will enable you to indulge in some very exotic fantasies of hypochondria. It will also serve to give you an idea about how you might build up a Knowledge Base in other fields and will demonstrate how the expert runs when it has a real slab of knowledge inside it.

Once you have loaded this try altering some of the items to see how it affects the expert's skills.

For instance, you might want the expert to take account of the fact that Chronic Bronchitis (illness 10) is more common in men than in women

('sympton' 53). So you could add to illness 10:-

53, .8, .5,

i.e., if the patient has chronic bronchitis then he is male with probability 0.8 - and, if the patient does not have chronic bronchitis then the probability of male is 0.5.

In other words, the healthy population is an even mixture of males and females.

Many diseases are sex-related in their incidence so question 53 could be added to a number of points in the Knowledge Base.

If you add question 53 to enough illnesses then you are likely to find that RV (53) rises and one of the questions the expert asks very early on in a consultation will be to determine the patient's sex. Which seems reasonable - few (human) doctors would be willing to carry out a diagnosis without this, very basic, item of knowledge.

Then try adding another illness - bubonic plague, for instance, and see if the expert can diagnose that. If you need to add more symptoms (questions) these can easily be tacked on to the end of the list and referenced by their number in the same way as existing symptoms.

Then try writing a Knowledge Base of your own with questions and 'illnesses' relating to the problem of why, for instance, your car won't start (it's probably got bubonic plague).

MEDICAL

```
1 REM :KNOWLEDGE BASE IN THE DOMAIN OF MEDICAL DIAGNOSIS TO BE DRIVEN BY A
BAYESIAN INFERENCING ENGINE



1000  DATA ILLNESSES
1010  DATA COMMON COLD,.02,1,.9,.05,2,.8,.02,3,.8,.02,5,.6,.01,6,1,.01,7,.2,.01,8,.5,.01,
      15,.8,.01,34,0,.01,999
1020  DATA ALLERGIC RHINITIS,.01,1,1,.01,2,1,.01,6,.9,.01,10,.7,.01,11,.7,.01,12,.6,.01,20,.9,
      .01,999
1030  DATA SINUSITIS,.01,14,.8,.01,13,.9,.01,15,.8,.01,7,.6,.01,22,.5,.01,2,.5,.001,6,.5,.01,63,
      .9,.01,999
1040  DATA PHARYNGITIS,.02,3,1,.01,16,.9,.01,8,.5,.01,11,.9,.01,37,.8,.3,64,.4,.01,999
1050  DATA TONSILLITIS,.001,3,1,.01,7,.9,.01,15,1,.01,16,.7,.01,19,0,.5,8,.8,.01,34,0,.01,64,.8,
      .01,999
```

```
1060 DATA INFLUENZA,.01,3,.9,.01,1,.9,.01,6,.5,.01,7,.7,.01,8,1,.01,15,1,.01,17,.8,.01,18,.6,
     .01,34,0,.01,999
1070 DATA LARYNGITIS,.01,4,1,.01,8,.6,.01,15,.05,.01,16,.7,.01,37,.8,.3,999
1080 DATA TUMOUR OF THE LARYNX,.00004,4,1,.01,34,.99,.01,37,.8,.3,999
1090 DATA ACUTE BRONCHITIS,.005,5,.1,.01,8,1,.01,12,1,.01,15,1,.01,18,.5,.01,21,1,.01,
     31,.9,.01,34,0,.01,22,.9,.01,999
1100 DATA CHRONIC BRONCHITIS,.005,5,1,.01,12,.9,.01,14,.5,.01,21,1,.01,22,.8,.01,34,1,
     .01,36,.9,.01,37,.8,.3,999
1110 DATA ASTHMA,.02,12,.8,.01,22,1,.01,23,.5,.01,24,.5,.01,25,.5,.01,26,.5,.01,31,.8,.01,999
1120 DATA EMPHYSEMA,.01,22,1,.01,5,.001,.01,26,.8,.01,12,.001,.01,21,.001,.01,37,.8,.3,999
1130 DATA PNEUMONIA,.003,8,1,.01,15,1,.01,18,.8,.01,22,1,.01,23,.5,.01,26,.5,.01,28,.1,
     .01,29,.02,.01,27,.2,.01,31,.9,.01,36,.1,.9,7,.9,.01,17,.9,.01,32,.5,.001,999
1140 DATA PLEURISY,.001,31,.8,.01,32,.8,.01,22,.5,.01,5,.8,.01,8,.9,.01,15,1,.01,34,0,.01,999
1150 DATA PNEUMOTHORAX,.0002,18,.8,.01,22,.8,.01,32,.8,.01,999
1160 DATA BRONCHIECTASIS,1E-5,21,1,.01,27,.5,.01,5,1,.01,14,.5,.01,999
1170 DATA LUNG ABSCESS,1E-5,33,.9,.01,18,.5,.01,21,.5,.01,27,.5,.01,999
1180 DATA PNEUMOCONIOSIS,.001,22,1,.01,36,1,.01,21,.8,.01,9,1,.001,999
1190 DATA LUNG CANCER,.001,5,1,.01,21,.8,.01,27,.5,.01,22,.5,.01,18,.8,.01,12,.5,.01,37,
     .99,.3,999
1200 DATA INTERSTITIAL FIBROSIS,1E-5,22,.8,.01,35,.8,.01,21,.6,.01,999
1210 DATA PULMONARY OEDEMA,.001,22,.9,.01,25,.9,.01,30,.5,.01,27,.5,.01,26,.5,.01,12,
     .8,.01,999
1220 DATA GASTRITIS,.01,41,.8,.01,43,.8,.01,42,.5,.01,8,.4,.01,37,.9,.5,999
1230 DATA HIATUS HERNIA,.001,18,.9,.01,32,.5,.001,42,.8,.001,57,.9,.01,16,.9,.01,41,.8,.01,999
1240 DATA DUODENAL ULCER,.01,37,.8,.2,42,.99,.001,41,.8,.01,999
1250 DATA PEPTIC ULCER,.01,42,.9,.001,18,.5,.01,20,.8,.01,41,.7,.01,56,.9,.01,62,.0001,.01,999
1260 DATA DIVERTICULAR DISEASE,.001,42,.6,.001,43,.5,.01,41,.5,.01,8,.5,.01,49,.5,.01,999
1280 DATA CROHN'S DISEASE,.0001,42,.9,.001,43,.9,.01,15,.9,.01,8,.7,.01,62,.00001,.01,999
1290 DATA INTESTINAL OBSTRUCTION,.00001,42,.9,.001,43,.8,.01,41,.5,.01,999
1300 DATA APPENDICITIS,.001,34,.1,.9,42,.9,.001,41,.8,.01,8,.8,.01,44,0,.5,999
1310 DATA FOOD POISONING,.001,42,.5,.001,41,.9,.01,43,.9,.01,7,.8,.01,999
1320 DATA GASTROENTERITIS,.01,41,.8,.01,42,.7,.001,43,.9,.01,8,.5,.01,999
1330 DATA KIDNEY STONES,.001,42,.7,.001,999
1340 DATA ACUTE PYELONEPHRITIS,.001,42,.9,.001,8,.8,.01,41,.7,.01,67,.9,.01,999
1350 DATA GALLSTONES,.01,42,.5,.001,41,.5,.01,57,.9,.01,999
1360 DATA CHOLECYSTITIS,.001,42,.8,.001,8,.9,.01,41,.8,.01,45,.8,.001,999
1370 DATA SHINGLES,.001,42,.5,.001,18,.5,.001,60,.9,.01,59,.9,.01,2,.6,.01,46,.5,.01,999
1380 DATA DEEP VEIN THROMBOSIS,.0005,40,.8,.01,999
1390 DATA RHEUMATOID ARTHRITIS,.001,15,.8,.01,17,.8,.01,40,.5,.001,999
1400 DATA HEART FAILURE,.001,22,.9,.01,36,.5,.01,25,.5,.001,12,.6,.01,18,.5,.001,32,.3,
     .001,40,.5,.01,42,.5,.01,28,.3,.001,47,.9,.01,999
1410 DATA ANXIETY,.01,46,.9,.01,28,.3,.01,47,.6,.01,39,.8,.01,23,.6,.01,48,.6,.01,16,.3,.01,
     43,.2,.01,22,.5,.01,50,.5,.01,57,.5,.01,58,.5,.01,15,.5,.01,7,.5,.01,4,.5,.01,999
1420 DATA DEPRESSION,.01,47,.5,.01,7,.5,.01,49,.5,.01,50,.5,.01,15,.5,.01,62,.8,.05,999
1430 DATA CORONARY THROMBOSIS,.01,18,.5,.01,32,.9,.001,20,.5,.01,36,0,.2,38,.5,.01,22,
     .5,.01,23,.5,.01,41,.5,.01,15,.9,.01,999
1440 DATA ANGINA,.01,37,.8,.3,18,.9,.01,36,.9,.01,22,.5,.01,23,.5,.01,38,.5,.01,41,.3,.01,999
1450 DATA PULMONARY EMBOLISM,.0001,22,1,.01,18,.7,.01,21,.6,.01,27,.5,.001,25,.5,
     .001,26,.4,.0001,999
1460 DATA STROKE,.001,28,.8,.01,38,.7,.01,51,.8,.001,58,.9,.01,61,.9,.01,999
```



```
1470 DATA TRANSIENT ISCHAEMIC ATTACK,.001,28,.8,.01,38,.7,.01,51,.8,.001,34,0,.01,20,
     .5,.01,58,.9,.01,61,.9,.01,999
1480 DATA TUBERCULOSIS,.0001,7,.5,.01,8,.5,.01,12,.5,.01,15,.5,.01,18,.5,.01,5,.5,.01,30,.5,
     .01,27,.5,.001,22,.5,.01,62,.0001,.01,23,.5,.01,999
1490 DATA HAEMORRHOIDS,.01,52,.9,.001,49,.8,.01,56,.9,.01,59,.5,.01,999
1500 DATA HYPOTHYROIDISM,.001,49,.8,.01,17,.5,.01,24,0,.01,23,.001,.01,39,.001,.01,4,
     .5,.01,43,0,.01,46,.001,.01,48,.001,.01,62,.9,.05,999
1510 DATA IRRITABLE COLON,.0007,43,.5,.01,49,.5,.01,42,.8,.001,41,.3,.01,57,.9,.01,999
1520 DATA CANCER OF LARGE INTESTINE,.001,43,.9,.01,49,.9,.01,52,.5,.001,42,.5,.001,56,
     .9,.01,62,.0001,.01,999
1530 DATA ULCERATIVE COLITIS,.0004,42,.8,.001,43,.8,.01,52,.6,.001,23,.5,.01,41,.5,.01,8,
     .5,.01,34,.4,.01,56,.9,.01,999
1540 DATA MENIERE'S DISEASE,.0005,38,.9,.001,41,.8,.01,34,.5,.01,20,.9,.01,999
1550 DATA CERVICAL SPONDYLOSIS,.006,54,.9,.01,7,.5,.01,38,.5,.01,58,.9,.01,61,.5,.01,999
1560 DATA SUBDURAL HAEMORRHAGE,.000001,55,.99,.0001,28,.9,.001,7,.9,.01,41,.9,.01,
     38,.9,.01,20,.5,.01,34,.5,.01,999
1570 DATA BRAIN TUMOUR,1E-6,7,.9,.01,41,.9,.01,38,.8,.01,50,.8,.01,34,.5,.01,999
1580 DATA MENINGITIS,1E-6,8,.9,.01,7,.9,.01,41,.9,.01,28,.7,.01,54,.9,.01,2,.9,.01,60,.5,.01,999
1590 DATA SUBARACHNOID HAEMORRAGE,.00001,7,.99,.01,54,.9,.01,38,.7,.01,28,.7,.01,
     41,.8,.01,2,.8,.01,999
1600 DATA ACUTE GLAUCOMA,.01,2,.9,.01,7,.9,.01,41,.7,.01,20,.8,.01,34,.8,.01,63,.8,.01,68,
     .9,.01,999
1610 DATA TEMPORAL ARTERITIS,.001,7,.9,.01,8,.7,.01,17,.7,.01,2,.8,.01,63,.99,.01,999
1620 DATA DYSPEPSIA,.1,18,.7,.01,57,.7,.01,42,.7,.01,41,.7,.01,46,.5,.01,20,.9,.01,999
1630 DATA HEART BLOCK,.0003,22,.5,.01,58,.8,.01,39,.6,.01,18,.6,.01,999
1640 DATA PERNICIOUS ANAEMIA,.0004,22,.9,.01,58,.9,.01,39,.9,.01,36,.9,.01,45,.5,.01,42,
     .5,.01,50,.5,.01,28,.4,.01,999
1650 DATA MIGRAINE,.1,7,1,.01,15,.9,.01,41,.9,.01,43,.5,.01,2,.9,.01,20,.9,.01,34,.9,.01,63,
     .99,.01,999
1660 DATA HYPERTENSION,.15,7,.5,.01,39,.5,.01,15,.5,.01,34,.9,.01,999
1670 DATA ECZEMA,.03,59,.9,.01,60,1,.01,999
1680 DATA URTICARIA,.03,59,.9,.01,60,1,.01,46,.5,.01,999
1690 DATA SCABIES,.001,59,1,.01,60,1,.01,999
1700 DATA MEASLES,.02,15,1,.01,8,1,.01,6,.9,.01,2,.9,.01,11,.9,.01,5,.9,.01,43,.5,.01,60,.8,
     .01,7,.5,.01,34,0,.01,999
1710 DATA RUBELLA,.01,8,.5,.01,60,.9,.01,54,.2,.01,34,0,.01,64,.5,.01,999
1720 DATA CHICKENPOX,.001,60,1,.01,59,1,.01,8,.8,.01,7,.5,.01,15,.5,.01,34,0,.01,999
1730 DATA PSORIASIS,.02,46,.6,.01,3,.5,.01,60,.99,.01,59,.5,.01,999
1740 DATA PITYRIASIS ROSEA,.01,60,1,.01,59,.9,.01,34,.5,.01,999
1750 DATA ACNE ROSACEA,.01,60,.9,.01,2,.5,.01,34,.8,.01,999
1760 DATA THYROTOXICOSIS,.001,46,.9,.01,47,.8,.01,48,.9,.01,23,.9,.01,39,.9,.01,22,.8,.01,
     43,.8,.01,62,.00001,.01,2,.5,.01,24,.9,.01,64,.3,.01,68,.3,.01,999
1770 DATA DIABETES MELLITUS,.01,62,.0001,.01,61,.5,.01,2,.5,.01,66,.99,.01,68,.1,.01,999
1780 DATA STOMACH CANCER,.0003,41,.5,.01,42,.7,.001,62,.0001,.01,52,.6,.001,56,.5,.01,999
1790 DATA ATRIAL FIBRILLATION,.001,39,.8,.01,38,.5,.01,42,.4,.01,58,.5,.01,999
1800 DATA HODGKIN'S DISEASE,.0001,23,.5,.01,63,.6,.01,54,.8,.01,59,.7,.01,64,.99,.01,999
1810 DATA GLANDULAR FEVER,.001,8,.9,.01,7,.9,.01,3,.9,.01,15,.9,.01,64,.8,.001,54,.8,.01,
     45,.5,.001,60,.5,.01,999
1820 DATA LYMPHOMA,.0001,64,.9,.01,54,.8,.01,15,.8,.01,62,.001,.01,8,.8,.01,23,.5,.01,59,
     .8,.01,999
1830 DATA MUMPS,.01,64,.99,.01,8,.8,.01,15,.9,.01,16,.7,.01,54,.6,.01,3,.8,.01,999
```

```
1840 DATA BELL'S PALSY,.0003,51,.9,.01,63,.5,.01,999
1850 DATA PARKINSON'S DISEASE,.001,48,.9,.01,51,.8,.01,42,.3,.01,50,.2,.01,28,.1,.01,999
1860 DATA RHEUMATIC FEVER,.01,3,.8,.01,15,.8,.01,8,.8,.01,64,.8,.01,60,.5,.01,59,.001,.01,
     48,.1,.01,999
1870 DATA CYSTITIS,.01,66,.9,.01,65,.9,.01,67,.9,.01,8,.5,.01,999
1880 DATA KIDNEY TUMOUR,.001,8,.6,.01,62,.0001,.01,41,.5,.01,42,.5,.01,65,.7,.01,999
1890 DATA BLADDER TUMOUR,.0004,65,.9,.01,42,.5,.01,66,.5,.01,67,.5,.01,8,.3,.01,999
1900 DATA IRITIS,.0005,2,.9,.01,68,.9,.01,999
1910 DATA ACUTE HEPATITIS,.001,8,.8,.01,15,.8,.01,17,.5,.01,42,.5,.01,45,.5,.01,41,.5,.01,999
2000 DATA SYMPTOMS
2010 DATA  1,ARE YOU SNEEZING A LOT ?
2020 DATA  2,ARE YOUR EYES PAINFUL OR WATERING A LOT ?
2030 DATA  3,DO YOU HAVE A SORE THROAT ?
2040 DATA  4,IS YOUR VOICE HOARSE ?
2050 DATA  5,ARE YOU COUGHING A LOT ?
2060 DATA  6,DO YOU HAVE A RUNNY NOSE ?
2070 DATA  7,DO YOU HAVE A HEADACHE OR - IN GENERAL - DO YOU SUFFER FROM
     HEADACHES AT ALL ?
2080 DATA  8,DO YOU HAVE A HIGH TEMPERATURE ? (OVER 100 F  SAY)
2090 DATA  9,DO YOU SPEND A LOT OF YOUR TIME IN A VERY DUSTY ATMOSPHERE ?
2100 DATA 10,DOES YOUR SKIN ITCH ?
2110 DATA 11, DO YOU HAVE A DRY THROAT ?
2120 DATA 12, IS YOUR BREATH 'WHEEZY' ?
2130 DATA 13,IS YOUR NOSE VERY 'BLOCKED UP' ?
2140 DATA 14,HAVE YOU HAD A COLD OR SIMILIAR INFECTION RECENTLY ?
2150 DATA 15,DO YOU FEEL GENERALLY ILL ?
2160 DATA 16,DO YOU HAVE TROUBLE SWALLOWING ?
2170 DATA 17,DO YOUR MUSCLES ACHE ?
2180 DATA 18,DO YOU HAVE ANY PAIN AT ALL IN YOUR CHEST ?
2190 DATA 19,HAVE YOU HAD YOUR TONSILS REMOVED ?
2200 DATA 20,DO YOU HAVE ANY SYMPTOMS WHICH TEND TO OCCUR IN 'ATTACKS'
     RATHER THAN BEING PRESENT ALL THE TIME ?
2210 DATA 21,DO YOU HAVE A 'PRODUCTIVE' COUGH - A COUGH IN WHICH YOU BRING
     SOMETHING UP ?
2220 DATA 22,ARE YOU RATHER BREATHLESS ?
2230 DATA 23,DO YOU SWEAT A LOT - NOT JUST WHEN YOU EXERT YOURSELF BUT WHEN
     YOU ARE APPARENTLY RELAXING AS WELL ?
2240 DATA 24,IS YOUR PULSE RATE HIGH ? NORMALLY IT SHOULD BE ABOUT 60 TO 80
     BEATS EACH MINUTE AND SLIGHTLY FASTER FOR PEOPLE OVER 70 OR UNDER 20
2250 DATA 25,DO YOU HAVE SEVERE ATTACKS OF BREATHLESSNESS - ENOUGH TO
     SERIOUSLY WORRY YOU ?
2260 DATA 26,DOES YOUR SKIN HAVE A BLUISH TINGE ?
2270 DATA 27,WHEN YOU COUGH IS YOUR PHLEGM STAINED WITH BLOOD ?
2280 DATA 28,ARE YOU CONFUSED - MUDDLED ABOUT WHAT'S GOING ON AROUND
     YOU ?
2290 DATA 29,ARE YOU (OR THE PATIENT) DELIRIOUS - TALKING INCOHERENTLY WITH
     POOR MUSCULAR COORDINATION ?
2300 DATA 30,DO YOU HAVE A DRY (NON-PRODUCTIVE) COUGH ?
2310 DATA 31,IS IT PAINFULL WHEN YOU BREATHE OR COUGH ?
2320 DATA 32,DO YOU EVER HAVE ANY REALLY SEVERE PAIN IN YOUR CHEST ?
2330 DATA 33,DO YOU SWING BETWEEN FEELING CHILLED AND FEELING FEVERISH ?
```



```
2340 DATA 34,DO YOU HAVE ANY SYMPTOMS WHICH HAVE BEEN PRESENT FOR SOME
     TIME - POSSIBLY SIX WEEKS OR MORE ?
2350 DATA 35,DO YOU HAVE 'CLUBBED FINGERS' ? - THESE ARE FINGERS IN WHICH THE
     CUTICLES HAVE ALMOST DISAPPEARED AND THE NAILS CURVE OVER AT THE FINGER
     TIPS
2360 DATA 36,DO YOU HAVE ANY SYMPTOMS WHICH MAINLY OCCUR WHEN YOU EXERT
     YOURSELF ?
2370 DATA 37,DO YOU SMOKE ? TO REPLY DIVIDE THE NUMBER OF CIGARETTES YOU
     SMOKE BY 5 - USE THIS NUMBER TO REPLY (IF YOU SMOKE 20 A DAY REPLY 4). 5 IS THE
     MAXIMUM REPLY AND -5 MEANS YOU DO NOT SMOKE.
2380 DATA 38,DO YOU SUFFER FROM FEELINGS OF DIZZINESS ?
2390 DATA 39,DO YOU HAVE PALPITATIONS ? - THE FEELING THAT YOUR HEART IS
     BEATING MORE STRONGLY OR FASTER OR MORE UNEVENLY THAN IT SHOULD.
2400 DATA 40,IS EITHER OF YOUR ANKLES UNDULY SWOLLEN ?
2410 DATA 41,ARE YOU VOMITING OR DO YOU HAVE STRONG FEELINGS OF NAUSEA ?
2420 DATA 42,DO YOU HAVE ANY ABDOMINAL PAIN ? THIS IS PAIN ANYWHERE BETWEEN
     THE BOTTOM OF THE RIBCAGE AND THE GROIN.
2430 DATA 43,DO YOU SUFFER FROM DIARRHOEA ? - PASSING UNUSUALLY RUNNY
     FAECES.
2440 DATA 44,HAVE YOU HAD YOUR APPENDIX REMOVED ?
2450 DATA 45,DO YOU HAVE JAUNDICE ? THIS IS NOT A DISEASE BUT A SYMPTOM OF
     DISEASE. OFTEN IT IS MOST OBVIOUS IN THE EYES - THE WHITES BECOME YELLOW.
2460 DATA 46,ARE YOU RATHER TENSE AND APPREHENSIVE ?
2470 DATA 47,DO YOU FIND IT HARD TO GET TO SLEEP OR DO YOU OFTEN WAKE IN THE
     MIDDLE OF THE NIGHT ?
2480 DATA 48,DO YOU HAVE ANY INVOLUNTARY TWITCHING OR TREMBLING ?
2490 DATA 49,DO YOU SUFFER FROM CONSTIPATION ? PASSING FAECES INFREQUENTLY
     OR WITH DIFFICULTY.
2500 DATA 50,DO YOU HAVE A POOR MEMORY ? THAT IS - DIFFICULTY IN REMEMBERING
     INDIVIDUAL FACTS EITHER OCCASIONALLY OR REGULARLY.
2510 DATA 51,HAVE YOU TOTALLY OR NEARLY LOST THE POWER OF SPEECH ?
2520 DATA 52,HAVE YOU EXPERIENCED ANY BLEEDING FROM YOUR BACK PASSAGE ?
2530 DATA 53,ARE YOU MALE OR FEMALE ? ANSWER 5 FOR MALE OR -5 FOR FEMALE. IF
     YOU'D LIKE THE ANALYSIS TO BE GENERAL (FOR EITHER SEX) REPLY 0.
2540 DATA 54,IS YOUR NECK STIFF AND/OR PAINFUL ?
2550 DATA 55,HAVE YOU SUSTAINED ANY KIND OF HEAD INJURY OVER THE LAST FEW
     WEEKS ? EVEN A VERY SLIGHT INJURY CAN BE IMPORTANT.
2560 DATA 56,HAVE YOU RECENTLY BEEN PASSING ABNORMAL-LOOKING FAECES ?
2570 DATA 57,ARE YOU PASSING LARGE QUANTITIES OF WIND - EITHER BY BELCHING OR
     FLATULENCE ?
2580 DATA 58,DO YOU HAVE SUDDEN FEELINGS OF FAINTNESS - FEELING WEAK AND
     UNSTEADY MAYBE EVEN LOSING CONSCIOUSNESS ?
2590 DATA 59,DOES ANY PART OF YOUR BODY ITCH - WITH OR WITHOUT THE PRESENCE
     OF ANY RASH ?
2600 DATA 60,DO YOU HAVE A SKIN RASH OF ANY SORT ?
2610 DATA 61,IS ANY PART OF YOUR BODY NUMB - OR DO YOU HAVE A TINGLING 'PINS
     AND NEEDLES' SENSATION ANYWHERE ?
2620 DATA 62,ARE YOU OVERWEIGHT OR UNDERWEIGHT ? REPLY 5 FOR DEFINITE
     OVERWEIGHT AND -5 FOR DEFINITE UNDERWEIGHT. REPLY 0 IF YOUR WEIGHT IS JUST
     RIGHT.
2630 DATA 63,DO YOU HAVE ANY PAIN IN YOUR FACE OR FOREHEAD ?
2640 DATA 64,DO YOU HAVE ANY SWELLINGS UNDER THE SKIN ?
```

```
2650  DATA 65,IS YOUR URINE ABNORMALLY COLOURED ?
2660  DATA 66,ARE YOU URINATING UNUSUALLY FREQUENTLY ?
2670  DATA 67,IS IT PAINFUL WHEN YOU URINATE ?
2680  DATA 68,IS YOUR VISION IMPAIRED IN ANY WAY - BLURRING OR DOUBLE VISION OR
      SEEING FLASHING LIGHTS ? (THIS DOES NOT INCLUDE DEFECTS WHICH CAN BE
      CORRECTED BY SPECTACLES).
3000  DATA END,END
```


Chapter 11

The Tower of Babel

Once upon a time, a long time ago, a group of computer scientists were sitting round a table deep in thought. And, in between thinking, they were counting how many transistors they had between them.

"Do you know," said one, "if we fastened *all* of these transistors together we'd have a computer that could do *anything.*" To which statement the others (in between swigs of beer) readily agreed. And, later that same evening, they got out their soldering irons and started work.

Time passed and the machine they were building got bigger and bigger until it was soon apparent that it, really and truly, would be able to carry out any task to which they set it. A more powerful computer had never been built and they dreamed of the day when they would switch it on and, in a matter of seconds, discover the complete answers to every problem in the Entire Known Universe.

As the machine rose skywards vast armies of programmers were recruited to write the programs that would control the new monster and each of these was initiated into the secret rites of machine code programming because, at that time, that was the only programming language that anyone had ever thought of.

The day of the great switch-on drew near and word of the great new computer spread far and wide causing not a little apprehension amongst the population as they thought of the awful cognitive power that was soon to be unleashed.

And, on the very night before the machine's completion, that self-same group of computer scientists again sat around a table, swigging beer, and debating which program should have the honour of being the first program to be written for the new machine.

"Y' know," said one sagely, "it doesn't really matter what program is first."

At which the others nodded their own heads wisely. "What really matters is the language it is written in."

At which point the others carried on nodding their heads and began to wonder what other languages there could possibly be, other than machine code.

But the speaker held his ground and pointed out that machine code is difficult to write and difficult to understand. It is machine-orientated, not orientated towards the problems that want solving. And, it had that greatest of all flaws - it was academically unsatisfying.

So, that night, each of the computer scientists went home and wrote a high level language definition and a compiler to go with it. And when the next day dawned bright and clear they reassembled themselves at the foot of the great machine and decided that, before they switched it on, they'd settle for once and for all, which was the best of the new languages to use. And, being gentlemen, they decided to settle the matter by discussion.

As far as is known they're still discussing the matter and the machine never got switched on, because the real problem was that none of them would ever understand the languages the others had developed.

And, even if that story isn't one hundred per cent historic fact, there's still a lot of truth in it.

For when the only language around was machine code (or Assembler, which is a bit easier to understand but, essentially, the same) then every computer person understood, more or less, what every other computer person was doing. There are now so many programming languages available that it's more or less impossible to be familiar with them all - yet if ideas are sketched out in an unfamiliar language you'll have difficulty in understanding those ideas. The language becomes a hindrance rather than a help.

That's why the examples given in this book are in BASIC - because it's a language that most people understand. But the real question is: Is BASIC the best language to use? Would it be better to have a crack at another language?

Intuitively, you might think there were strong arguments for using, say, LISP because that's the language in which most 'professional' expert systems are written. Or, maybe, Prolog - a derivative of LISP.

To get at this question we need to consider just what a programming

language can and cannot do - and the first point to note is that it really and truly can't do anything which can't be done in machine code. For most people this might be obvious but it's still worth noting that any language is always compiled down into machine code prior to execution and it is the machine code which defines the machine on which the program is run. If an instruction cannot be reduced to machine code then, on that machine, it is unexecutable. So, in machine code it's possible to do everything of which the machine is capable and, in any other language it is impossible to do any more than that.

So: if a language cannot do more than machine code can do, can it be such that it prevents the programmer from doing as much as is possible in machine code? That is: can a language reduce the range of possible activities? The answer is: Yes. And, for an example, consider BASIC and pretend that there aren't any such things as PEEK and POKE. Now suppose that, for some reason of your own you wanted to look into byte 3096 in RAM, see what was there, and then alter only one bit in that byte. That would be impossible except by using the weirdest contortions imaginable because BASIC gives you no way (without PEEK or POKE) of accessing an absolute address and no way of 'bit-twiddling'.

In fact, this example isn't particularly unusual. Many high level languages don't give much scope for absolute addressing and yet there are plenty of applications for which absolute addressing is essential. Real-time programming gives plenty of examples of this - if the computer had to read a measurement from an external meter, say, then almost always the programming would have to rely on absolute addresses to get the meter reading into memory.

So a language can never do more than machine code and can, often, do much less. In which case, what are other languages good for?

Two things: One, they are easier to use than machine code. Two, they can (but don't always) act as an aid to thought.

The first point is obvious. BASIC is easier than machine code both to learn and to use. That's true of most high level languages.

But the matter of an aid to thought isn't quite so obvious and, even when it is, it can be rather a two-edged sword. Staying with BASIC consider the FOR-loop. That is certainly an aid to thought. You can conjure it up in your head, look at it on paper, display it on the screen and it's quite clear what it does. So you can forget it and concentrate on what's inside the loop. At which point the second edge of the sword starts to appear - because we rather presupposed there that, with BASIC, you are definitely going to use

FOR-loops. In fact, such a strong aid to thought is the FOR-loop that, when faced with a programming problem, the temptation is to try to work out how best to arrange FOR-loops to solve that problem rather than to ask whether or not FOR-loops are, in fact, a good way to proceed in this instance. And if they weren't a good way to proceed then the odds are that you'd still use them anyway.

To make this point clearer consider the dichotomy between the computer and the problem the computer has to solve.

Machine code is exclusively computer-orientated. If you simply want to drive the machine then machine code is the most efficient way of programming in terms of getting a short, fast program.

But, to a greater or lesser degree, high level languages are problem orientated - they look after the machine and let you concentrate on solving the problem. And with, say, BASIC if you want to work on a matrix of numbers then FOR-loops provide an ideal way of accessing the various elements of those matrices so that you can do so. The mistake to avoid though is to think that because a language is problem orientated it is ideally orientated towards any problem that you might want to solve. There are different types of problems which can be solved in different types of ways - as the names of some of the programming languages reveal. COBOL, for instance, is an acronym for Common Business Oriented (sic) Language - it's aimed at business problems. ALGOL is an acronym for Algorithmic Language - and it's aimed at problems for which an algorithmic solution is handy (an algorithm is a recipe for solving problems primarily in the field of logic and maths). FORTRAN is an acronym for Formula Translator - a language designed for evaluating formulae. And BASIC? That's Beginners All-Purpose Symbolic Instruction Code - a language designed, actually, for the problem of beginners who don't know how to program at all well!

Back to expert systems though and the question: Is there a language aimed at these?

Obviously, to answer that question you need to know the sort of problems that arise when you're building an expert system. And that's difficult to specify simply because of the wide variety of problems which are subsumed under the heading 'expert systems'.

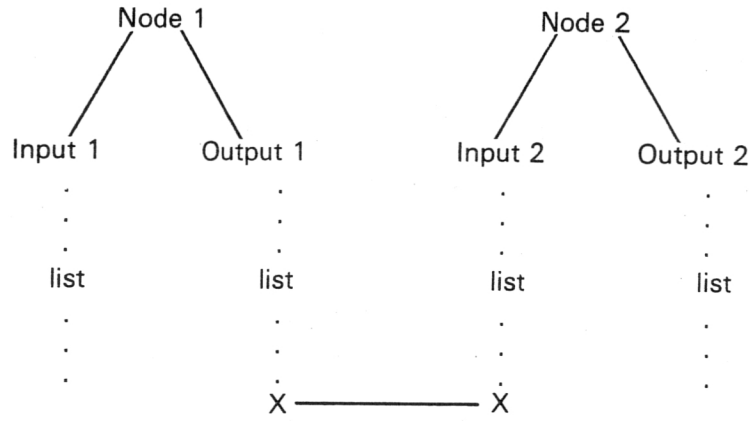
However, there are a few points which can be made. For a start think of that multi-node expert system we dreamed up. In BASIC the nodes were connected to each other by means of labelling them with a string variable and then searching through all the string variables to see if there was a

matching string to determine the connections. Now that was pretty messy - and it wouldn't be helped any by the fact that those variables are data and will be likely to be lost as soon as you switch the machine off (or the power fails).

This is the sort of problem that LISP was designed to get around for what we have is a series of lists (LISt Processing Language, you see!) Suppose now, for example, we have two nodes, node 1 and node 2 with inputs and outputs to each.

Define a list, Node 2. This list contains all the inputs and outputs to that node. Define two sub-lists in Node 2, Input and Output, to show which is which. Do the same for Node 1. Now, if Node 1 has an Output 1 which contains X which is the same element as node 2 Input 2 then it's possible to define that in a LISP program also.

We can define a structure something like this:



And all of this is contained in the program code - not as data as such (although, to be honest, the distinction between program and data isn't very strong in LISP anyway).

Having defined the above structure it is fairly easy then to write code to ask if any of the inputs on Node 2 are outputs from Node 1. That's a fairly

simple example and the real value of LISP is that it allows these structures to be defined in as complex a fashion as you like. You can just go on and on making things more and more complicated - something which would make the average head hurt if it was tried in BASIC.

Consider a further point - that all of the systems we have given have been forward chaining. They collected items of data and then worked forward through the program to see what could be done with that data. Suppose that we'd wanted a backward chaining program? In the above example the distinction goes like this:

In forward chaining we provide the program with, say, Input 1 on Node 1 which produces output which is passed to Node 2 as input so that Node 2 can produce output.

In backward chaining the program looks at Node 2, thinks it will see if it can produce a specific output, looks backwards to see what inputs it needs, sees Node 1 is needed for some of these inputs and then goes back to Node 1 and asks for input to that. Not very easy in BASIC. The main reason why it's hard is because most BASICs don't offer recursion - the ability of a procedure to call itself.

The most usual example of recursion (in case you're not sure what it is) is the evaluation of a factorial - say, 3!=3*2*1=6. So, if we had the recursive procedure FAC(N) we could say that

```
FAC(N)   = N*FAC(N-1) if N is greater than 1
         = 1 if N less than or equal to 1
```

So

```
FAC(3) = 3*FAC(2) =3*2*FAC(1) = 3*2*1 = 6.
```

It isn't always so easy to give examples for less simple matters but suppose we had a recursive procedure NODE(N). This procedure tries to evaluate Node n and does it by considering the inputs to that node. While it is considering these inputs it also checks to see if any of them are the outputs from Node (n-1) and, if they are, it calls NODE(N-1). Which then checks Node (n-2) and so on until the method has got back to the front of the chain.

Now, that could be useful. Writing NODE(N) might take some thought but, once written, it could be used on the arbitrarily complex structures we mentioned earlier. But, of course, the fact that most BASICs don't support recursion doesn't completely rule out recursive code. It would be possible to write BASIC code which did behave like this - but it would be fiddly.

Using LISP (or Prolog) makes life easier if recursive activities seem like a good idea for solving the problem in hand.

In a way, the advantages of these languages lie largely in the fact that they are organised to help deal with very complicated structures which, largely, have very little to do with ordinary mathematics. And the application to expert systems lies in the fact that, when we try to make our expert system behave like a human expert we are often trying to make it work with a large amount of rather complicated, non-numeric, structures.

The point about non-numeric or non-mathematical structures helps to display another side to the coin. The fact that if you need to do maths these languages can be rather laborious. Take for instance the BASIC statement X=2+2 and consider the LISP alternative which is (SETQ X ADD(2 2)). Not too complicated, but consider what a real calculation might look like!

What's happened is that LISP was designed to help solve a particular type of problem - and it was not the type of problem associated with maths expressions. But some LISP users might want to do a bit of maths so that side of it was expressed in just the same way as the rest. With the result that the maths is somewhat on the weird side! In BASIC we find the opposite. Beginners maybe did want to do lots of maths - so design a language aimed at those problems. And, of course, you find that the language so designed is somewhat weak in other areas.

What we really want is one, big, language that does everything under all circumstances, but we'll be unlikely to get it simply because nobody knows what it might look like. Even our very own English isn't up to every task - witness the large number of specialist sub-languages that have been evolved over the years to extend English into new fields - like, for instance, that of the mathematician or the doctor. If we wish to produce a language that would be perfect for, say, medical diagnosis we would need English (to communicate with the patients), Doctor-ese (to communicate with the medics - maybe Latin would do), and Mathematics (to work out any calculations). And there's no harm in saying that this is rather more than many practising physicians have!

In a way the problem of languages isn't really so bad as might be made out for the simple reason that most computer users only have access to a limited number of choices - and that settles most arguments about which language should be used. There's also the fact that, often, the best language to use is the language you're good at. A great deal of research in artificial intelligence has relied on LISP, for instance, but one reason for this is that students of artificial intelligence are often taught LISP and, after that, the habit is bound to die just as hard as does the BASIC-habit in someone else.

But certainly the language you use influences, one way or another, the way you think about problems and the BASIC programmer is likely to find himself implementing an expert system in quite a different way to the way a LISP programmer might do it.

In the end the value of each approach lies simply in the quality of the end result and, in the final extreme, you can always go back to machine code. The only danger of which is that you might die of old age before you finish writing the program.

Chapter 12

Summary and Technical Overview

As this book reads rather like a story, the plot of which involves getting you to do something, you may well be sitting there and saying something like:

"Fine, I can see how these systems work. I could even write my own expert now but I can't remember how this Naylor character defined 'rules' or ..."

Well, you could look it up in the index or consult those notes you've been making. But, to make it really easy, here is a list of the important terms that you'll need to know:

12.1 Events

An Event can be almost anything.
Let H be the event that a given hypothesis is true.
Let E be the event that a given piece of evidence occurs which may, or may not, support the given hypothesis.

12.2 Probabilities (see also pages 20 - 26)

P(H) is the probability that H is true.
P(E) is the probability that E occurs.
Probabilities are numbers in the range 0 to 1.
If the probability is 0 then the event never occurs.
If the probability is 1 then the event always occurs.
P(not H) = 1 - P(H) and is the probability that H does not occur.

Two events are independent if their joint probability of occurence equals the product of their separate probabilities of occurence.

$P(E_1 \& E_2)$ is the joint probability that both E_1 and E_2 occur.
E_1 and E_2 are independent if, and only if, $P(E_1 \& E_2) = P(E_1)\ P(E_2)$.

P(H:E) is the conditional probability of H occuring given that E has already occurred.
If H and E are independent P(H:E) = P(H).

In general: $P(H:E) = \frac{P(H\&E)}{P(E)}$

Similarily, $P(E:H) = \frac{P(E\&H)}{P(H)}$

So: $P(H:E) = \frac{P(E:H)P(H)}{P(E)}$

12.2.1 Bayes' Theorem (see also pages 33, 190 - 195, 201 - 205)

$$P(H:E) = \frac{P(E:H)P(H)}{P(E:H)P(H) + P(E:\text{ not } H)P(\text{not } H)}$$

P(H:E) versus P(E:H):-

It might, initially, seem much more sensible to work always with P(H:E) - after all, we want to know what the probability is for each hypothesis given the evidence, not the other way round.

The problem is that P(H:E) is not at all an obvious quantity. If you knew P(H:E) to start with there would be very little point in writing a computer program to do it for you.

P(E:H), on the other hand, is usually much more apparent when data is being collected on the problem in hand.

You ask the question: If H is true, what is the probability of observing this particular piece of evidence?

And, for P(E:not H): If H is not true, what is the probability of observing this particular piece of evidence?

These two questions enable the program to calculate P(H:E) successively improving the estimate as each new piece of evidence comes in.

For two, or more, events E_1 and E_2:

If E_1 and E_2 are independent:

$$P(E_1 \& E_2 : H) = P(E_1 : H)P(E_2 : H)$$

Note: it is never correct to say:

$$P(H : E_1 \& E_2) = P(H : E_1)P(H : E_2)$$

If E is the event' all E_i occur', and the E_i are independent of each other, then we can calculate:

$$P(E:H) = P(E_1:H)P(E_2:H)...P(E_n:H)$$
$$P(\text{not } E:H) = P(\text{not } E_1:H)P(\text{not } E_2:H)...P(\text{not } E_n:H)$$

12.2.2. Prior and posterior probabilities

Suppose we have a hypothesis H and some evidence, for or against H, which we call E.

Then:

P(H) is the prior probability of H. It is the probability of H with no knowledge of E.

P(H:E), or P(H:not E), is the posterior probability of H. It is the probability of H once we know the truth about E.

From Bayes' Theorem, we have:

$$P(H:E) = \frac{P(E:H)P(H)}{P(E:H)P(H) + P(E:\text{not } H)P(\text{not } H)}$$

Or, if E were found to be absent, we would have:

$$P(H:\text{not } E) = \frac{P(\text{not } E:H)P(H)}{P(\text{not } E:H)P(H) + P(\text{not } E:\text{not } H)P(\text{not } H)}$$

Suppose that, for some H, there are a large number of items of evidence which become progressively available to either support H or deny H. Call them E_1 to E_n.

If they were all available at once, and if they were all independent of each other, we could calculate P(E:H) as the product of the individual $P(E_i:H)$ and then find P(H:E) where E is the event 'all E_i occur'. Similarly, we could calculate P(not E:H) as the product of all the $P(\text{not } E_i:H)$.

But instead, we might find it more convenient to work in stages, totting up the evidence and its effects as we go through the E_i. We can do this using prior and posterior probabilities in the following way:

1. P(H) is the prior probability of H
2. For given evidence E_i we have $P(E_i:H)$ and $P(E_i:\text{not } H)$.
3. Using Bayes' Theorem we can calculate $P(H:E_i)$ or $P(H:\text{not } E_i)$ depending on the outcome of E_i. This is the posterior probability of H.
4. We can now disregard E_i altogether and call the posterior probability of H the (new) prior probability of H. So: put $P(H) = P(H:E_i)$, or $P(H:\text{not } E_i)$ depending on the value of E_i.
5. Select a new E_i to consider. Go to 1.

12.2.3 Odds

The odds in favour of an event may be calculated from the probabilities of that event:

$$O(E) = \frac{P(E)}{1 - P(E)}$$

And:

$$P(E) = \frac{O(E)}{1 + O(E)}$$

12.2.4 Approximations (see also pages 190 - 195)

$$P(A \text{ AND } B) = \min(P(A),P(B))$$
$$P(A \text{ OR } B) = \max(P(A),P(B))$$

These results are not strictly true and the extent to which they are in error depends on the extent to which A and B are independent or otherwise. If

information concerning the independence is not available they may, however, prove useful.

$$\min(x,y) = x, \text{ if } x \text{ is less than } y$$
$$= y, \text{ if } y \text{ is less than } x$$
$$\max(x,y) = x, \text{ if } x \text{ is greater than } y$$
$$= y, \text{ if } y \text{ is greater than } x.$$

12.2.5 Combinations (see also pages 31 - 32)

Given n events, of which we choose x, there are $\binom{n}{x}$

ways of making the selection.

$$\binom{n}{x} = \frac{n!}{(n-x)!x!}$$

where $n! = n(n-1)(n-2)(n-3)...(n-(n-1))$

e.g. $4! = 4.3.2.1. = 24$

If there are n items of evidence and any or all of these items may occur then there are:

$$\sum_{x=o}^{x=n} \binom{n}{x}$$

possible combinations of the evidence.

12.2.6 Descriptive statistics (see also pages 119 - 124)

Descriptive statistics are, as the name implies, descriptions usually made to summarise the main features of some data which we have.

Mean m

$$m = \sum_{i=1}^{i=n} \frac{x_i}{n}$$, often referred to as the 'average'.

Variance, v

$$v = \sum_{i=1}^{i=n} \frac{(x_i - m)^2}{n}$$

Standard Deviation, sd

sd = SQR(v)

12.2.7 Normal distribution

Most of the methods in this book involve the use of non-parametric statistics. Non-parametric statistics make no assumption about the underlying behaviour of the variables being studied and are, consequently, less prone to let you down.

Parametric statistics assume an underlying mathematical model for the behaviour of the variables being studied. One such family of parametric statistics is that group of statistics which assumes that variables basically come from a population which possesses the normal probability distribution function. This is the, well-known, bell-shaped curve. A large body of theory is available for handling normal distributions and it is frequently tempting to use normal distribution theory.

Before doing so it is essential to examine any data to see if it really is normally distributed. In practice, as opposed to theory, many variables are not normally distributed even though many statisticians wish they were. You should vigorously resist the temptation to apply normal distribution theory on variables which are not normally distributed.

12.2.8 Discrete and continuous variables

A discrete variable is one which can only adopt certain fixed values - for instance, Yes/No responses are discrete. Continuous variables are those which can adopt any number as their value - for instance, rainfall figures are continuous. Continuous variables may be made discrete by chopping them up into sections - for example, rainfall over one inch/rainfall under one inch.

But, you are likely to get into trouble if you use the same methods of working for both discrete and continuous variables together. Decide which you are using at the start and stick to that type of variable. In general, you will find discrete variables much easier to handle than continuous variables. To some extent this is because continuous variables tend to rely on parametric statistics - and the choice of suitable parametric families can be a vexed question.

12.3 Surfaces (see also pages 45 - 49)

The general equation of a surface is:

$$y = \sum_{i=0}^{i=n} b_i x_i$$

where the b_i are constants and the x_i variables.

A table top has a two-dimensional surface (n=2) and x_1 and x_2 are at right angles to each other. This can be expressed as saying that x_1 and x_2 are independent of each other, or are not correlated with each other. In general, there may be as many variables as you wish. They may be independent of each other, or not. They may be used to describe something which does not exist in real, three-dimensional, space. They may be a description, for instance, of a particular pattern of behaviour, or of disease, or of anything.

A surface is a mathematical convenience.

12.4 Discrimination (see also pages 45, 58, 142)

Suppose that we have n categories of object described by measurements on a certain number of variables. We are then given a further object and a set of measurements for this object on these variables. We then have to decide into which of the n categories it falls. This is the problem of discrimination.

In general, it is only possible to solve the problem completely if the various categories are linearly separable. That is to say, if it is possible to place a surface (or a series of surfaces) between each of the categories. The surfaces are defined in terms of the variables we are able to measure on these objects.

We may decide that the categories are mutually exclusive - that is to say, the object can fall only into one category of the n. In this case we place it in the most likely category.

This might mean, that category for which P(H:E) is a maximum. Here H defines one category and E is all the evidence relating objects to category membership.

Note that, mathematically, P(H:E) is the equation of a surface. It is a surface which 'points' in the direction of a particular H. The calculations for P(H:E) give, indirectly, the b_i for the surface. And the evidence E on each variable gives the x_i.

We do not, however, have to use probability measures explicitly. Any method that produces a discriminating surface will do. Any method that works is right.

If the categories are not mutually exclusive we may place our object into more than one category. In this case we do not simply choose the most likely category. We have a threshold criterion and the object is placed into those categories for which it exceeds the threshold criterion.

If the discriminating surface is

$$y = \sum_{i=0}^{i=n} b_i x_i$$

we can categorise the object for each case in which y exceeds some y_c.

12.5 The learning algorithm (see also pages 37, 50, 55, 61)

1. Take n observations x_i on an object to be categorised.

2. Calculate $$y_k = \sum_{i=0}^{i=n} b_{ik} x_i$$

for each of the possible categories. Initially, $b_{ik} = 0$ for all i,k.

3. Find that category k for which y_k is the greatest.
4. If the object belongs to category k then this categorisation is correct. No changes are necessary. Go to item 6.
5. If the categorisation is wrong, modify b_{ik} as follows:
   $b_{ik} = b_{ik} + x_i$ for that category k to which the object should have been categorised. Do this for all b_{ik} in k.

   $b_{ik} = b_{ik} - x_i$ for all categories k to which the object should not have been categorised and for which y_k is greater than the y_k for the correct classification. Do this for all b_{ik} in these incorrect catagories.
6. Take another observation. Go to 1.

This algorithm requires a training set of objects with known categorisations to get it going. Once it is working well it can be used on further objects whose correct classification is not known.

12.6 Parallel and sequential procedures (see also page 65)

A parallel procedure takes all the available information at once and makes one, final, calculation on the basis of this information.

$$y_k = \sum_{i=0}^{i=n} b_{ik} x_i \qquad \text{for all k categories}$$

A sequential procedure steps through the variables one by one making what use it can of the information as it goes along.

$y_{ik} = y_{i-1,k} + b_{ik}x_i$

A sequential procedure should, once all of the information has been collected, give the same result as a parallel procedure. If all of the information is always needed to obtain a conclusion then the differences between the two methods are purely cosmetic.

If a conclusion can be reached on the basis of less than all the possible information a sequential procedure could be more economical because it would come to a conclusion faster.

12.7 Minimum and maximum values (see also pages 70 and 78)

If the variables x_i which are to be provided to the system have minimum and maximum values associated with them then the procedure may be able to come to a conclusion more quickly.

Suppose that all max(x_i) always support a particular categorisation and that all min (x_i) always argue against such a categorisation. Then, for as yet unknown variables x_i calculate two possible outcomes based on max(x_i) and min (x_i) for the outstanding variables. If neither of these overturn the current 'best guess' of the system then that best guess cannot be overturned and can be taken to be correct. Therefore, the outstanding x_i are not actually needed.

12.8 Processing strategies (see also pages 137, 182, 183)

If all the x_i are required to make a categorisation then any difference in processing strategies is purely cosmetic. If not all x_i are needed then it becomes important to use the most efficient processing strategy possible in order to come to a conclusion as efficiently as possible.

12.8.1 Goal-driven strategies

These work by selecting a categorisation and, staying with that categorisation, checking out the appropriate x_i until it is possible to make a

decision about that particular category.

Having done that, the system can then proceed to check out the next category. And so on.

12.8.2 Data-driven strategies

These work by selecting an x_i which, on some grounds, looks like a useful x_i to know. Having got a value for this x_i the system makes what use it can of the information.

Having done that, the system goes on to select another x_i. In the course of this, conclusions are reached about various categorisations.

12.8.3 Selecting the next variable (see also pages 74 ,204)

Whether the system works on a goal-driven strategy or a data-driven strategy there will usually be some latitude with respect to which x_i should be examined next. The problem is one of choosing a 'good' question to ask from a series of possible questions and it is hard to be very precise about what constitutes 'good'. However, of any question, we may consider:

1. How many categorisations does it influence?
2. To what extent does it influence those categorisations?
   For example:
   1. For each currently unknown x_i we might calculate:
      y_{ik} (using the max x_i) - y_{ik} (using the min x_i)
      to determine the maximum possible change that can be brought about by a knowledge of x_i.
   2. Using Bayesian probabilities we might calculate:
      P(H:E) - P(H:not E)
      where E is the observation x_i, to determine the maximum change that can be brought about by each piece of evidence.

To work out these items for each variable prior to asking a question involves far more processing (on the part of the computer) than simply to ask the next question on a list. The advantage is that it might require, overall, less effort on the part of the person using that computer.

However, as short cuts to speed processing time:
Try: max(max(x_i)- min(x_i)) to give an idea of an 'important' question. The snag with this method is that x_i might have a very wide range of variation but might not be, actually, important in any way.

Try: max (variance (b_{ik})) to give an idea of the extent to which a variable is used in the categorisations.

The snag is that a variable which appears to be widely used might, in fact, have a small (max(x_i) - min(x_i)) and the large range in the b_{ik} might simply be to allow for this fact.

12.9 Intermediate conclusions (see also page 130)

Intermediate conclusions aren't always strictly necessary but they can be useful in 'humanising' the system. An intermediate conclusion can be used as an input variable to another stage in the expert process. The use of intermediate conclusions greatly increases the difficulty of writing the system in the first place.

However, there is one point which is worth bearing in mind with respect to intermediate conclusions - and this hinges on the rather vexed question of independence.

Most of the statistical methods used assume independence of the various items of evidence E_i and, frequently, the assumptions of independence are not justified.

If there are n items of evidence supporting a hypothesis H and they are correlated with each other but the calculations are made as if the correlations did not exist then H will receive more, apparent, support than it in fact deserves.

Inserting intermediate conclusions in the reasoning process can help to eliminate this effect.

Consider E_1 and E_2 both present to support H_1.

Let H_1 be an intermediate conclusion which acts as evidence for a further conclusion H_2.

Let E_1 and E_2 be, to some extent, correlated with each other.

Then H_1 receives more support than it should.

However, it might still be reasonable to claim H_1 to be true so no real harm is done at this stage.

But, without H_1 the error in calculating for E_1 and E_2 would be carried forward into H_2, gradually getting more and more serious as more and more, possibly correlated E_i are added into the calculations.

The presence of H_1 and other intermediate conclusions allows the slate to be wiped clean, as it were. A new set of calculations can be started for H_2 based on a smaller number of items of evidence and reducing the risk of a serious build up of uncertainty.

12.9.1 Explanatory systems (see also page 175)

Intermediate conclusions can be used to give explanations of the current state of the system.

In general, an explanatory system can say where it has got to in its reasoning process and, often, how it got there. The problem is that this tends to force the computer to work in a way which is capable of ready explanation and this may be difficult to achieve or may even lead to a less efficient implementation.

For instance, what sort of explanation would be appropriate if the computer asked for x_i because it had calculated that this particular x_i would influence P(H:E) more than any other x_i?

Most of the strategies in this book for selecting a 'good' x_i consist of finding an x_i which looks kind of important from a mathematical point of view. But a naive user might well feel happier with explanations that involved, say, a listing of which categorisations would be affected most by that x_i irrespective of why that particular x_i was really chosen.

12.10 Linear interpolation of responses (see also pages 195, 204)

Suppose that the user is not certain about his answer to a question. He may wish to reply on a certainty scale from, say, minus 5 to plus 5 corresponding to No (-5), Don't Know (0), and Yes (+5).

Suppose now that the prior probability of any response is P(E).

If there is uncertainty in the answer then the system must deal with this by putting a value on P(E) and, also, a value on P(not E). If the user is uncertain about the existence of a piece of evidence then that evidence might not be there.

Let the user's response be R.
If R is greater than, or equal to, 0 then
P(E) = P(E) + (1-P(E))R/5

If R is less than, or equal to, 0 then
P(E) = P(E) + P(E)R/5

Obviously, P(not E) = 1 - P(E)

and, if R=0 then P(E) remains unchanged at P(E).

Then, to calculate the new P(H:E) we have:

P(H:E) = P(H:E)P(E) + P(H:not E)P(not E)

In other words, calculate both outcomes and weight the final outcome by the certainty the user expressed for the evidence and against the evidence.

12.11 Data formats

1. Arrays

DIM R(E,H)

	Hypothesis 1	Hypothesis 2 ... etc.
Evidence 1	x	x
Evidence 2	x	x
..., etc.		

2. DATA statements (see also Section 10.1)

DATA hypothesis 1, prior probability of hypothesis 1, evidence j, $P(E_j:H)$, $P(E_j:\text{not } H)$

DATA hypothesis 2, etc.

and

DATA j, evidence j

The array R holds a series of equations for surfaces enabling a categorisation process to be carried out.

The DATA statements hold similar information - each DATA statement is very much like one column from the array.

The advantage of using arrays is that processing is quicker - but the data can be lost if you don't store it.

The advantage of using DATA statements is that they are less easily lost but can be slow to process.

A good method might involve holding the information on a disc file if you have one available.

Chapter 13

Select Readings

It's quite a problem producing a list of suggested reading for a subject as diffuse as expert systems because one of the best things to do is to read up on the subject in which you want your system to be expert - and that covers, potentially, everything.

Also, if the list is too long then nobody will ever start to look at the items it contains; and, if it's too short, then it will miss out a great deal. However (in alphabetical order)...

Bailey, N T J. Mathematics, Statistics and Systems for Health. Wiley, 1977.

This has a nice section on medical diagnosis. Because of the medical orientation it doesn't confine itself to theoretically 'perfect' methods but, instead, concentrates on methods that work.

Higman, B. A Comparative Study of Programming Languages. McDonald/Elsevier Computer Monographs, 1967.

A relatively cheap book, this will be a bit of an eye-opener to anyone who thought there were only three or four different programming languages. Like most books which cover a large number of languages this can cause mental indigestion if you try to read it too quickly.

Hunt, E B. Artificial Intelligence. Academic Press, 1975

A nice book on AI, though it might seem a bit abstract for the non-

mathematical reader in places. It covers the learning algorithm and Bayesian inferencing and is generally interesting on a wide range of subjects.

Kendall, M. Multivariate Analysis. Charles Griffin, 1975.

This is, actually, one of this author's all-time favourites.

It's a book on statistics and covers the problems of classification and distance. It isn't the most up to date book by any means - in fact, with reference to computers, it several times suggests that the use of VDUs (CRTs) could become widespread one day - but it contains a quantity of good, sound commonsense which you won't find anywhere else on this subject.

Knuth, D E. The Art of Computer Programming, Volume 1 Fundamental Algorithms. Addison-Wesley, 1973.

It's pretty well impossible not to have a copy of this. You have to have a copy it you want to mess around with linkages and tree structures although its big disadvantage is that all the examples are given in MIX - a sort of assembler-level language for a fictitous machine.

Michie, D (ed.). Expert Systems in the Microelectronic Age. Edinburgh University Press, 1979.

A useful collection of current papers on various systems, such as PROSPECTOR, MYCIN, PUFF, SU/X, MECHO and others. It's especially useful in revealing just how complicated things can get.

Mood, A.M, Graybill, F.A, Boes, D.C. Introduction to the Theory of Statistics. McGraw-Hill, 1974.

So many aspects of expert systems are tied in with statistics that the odd book on the subject is essential. The snag is, of course, that such books

can't always confine themselves simply to what you need to know, but give you an entire course in all sorts of statistics. However, you'll find Bayesian inferencing and basic probabilities covered here.

Morrison, D.F. Multivariate Statistical Methods. McGraw-Hill, 1976.

An alternative to Kendall. Rather a dry book but it uses matrix notation in contrast to Kendall's use of subscripted variables.
Some may find the one easier than the other (in programming languages they both refer to arrays).

Robinson, J.A. Logic: Form and Function. Edinburgh University Press, 1979.

Some people may find this a difficult book if they don't have too much training in logic but it does have a very nice section on LISP and, if you feel like moving on to more complex systems, you're going to have a hard job avoiding formal logic anyway.

Wani, J.K. Probability and Statistical Inference. Appleton-Century-Crofts, 1971.

Another book on statistics this again gives you Bayes and probabilities.

If possible, it's always a good idea to have at least two books on any given subject - especially if the subject is a difficult one.

That way you can read one version, fail to understand it, read another version, fail to understand that, then go back to the first version - at about which point the light often dawns.

In which case, as you've got one book now in your hand (this one) what is the other book you should rush out and buy?

Kendall probably. It's not a computer book as such - but it's a wonderful source of ideas for someone with a computer. Or, failing that, Hunt.

Index

Adaptive Systems(see also Learning Algorithm) 11
AGE 198
AI/X 198
Antecedent-consequent 184
Applications 164

Backward chaining 171 183
BASIC 180, 222
Bayes' Theorem 33, 193, 230
Belief 172

CASNET 197
Certainty Factor 172
Chi-square 96, 98
Classification 45, 50, 117, 236
Combinations 32, 233
Conditional probabilities 25, 26, 230
Continuous variables 235

Description Space 45, 50
DENDRAL 185
Discrete variables 235
Discrimination 50, 236
Distance measures 52, 117
Domain of enquiry 14

EL 197
EMYCIN 178
EXPERT 198

Factorials 32, 102
Forward chaining 138, 182

GUIDON 177

HEARSAY 198

IF... THEN 12, 171, 180
Independent Events 31, 231
INTERNIST 197

Joint probabilities 25, 30, 229

KAS 198
Knowledge base 12, 14, 146

Learning algorithm 37, 50, 55, 61, 237
Linear seperability 59
LISP 222
Logical connectives 144
Logical probability 171

Machine code 222
Maximum values 70, 78, 238
MECHO 197
Mean 52, 234
META-DENDRAL 188
Micro-Expert 198
Minimum values 70, 78, 238
MYCIN 170
MOLGEN 197

Normal distribution 120, 234

Odds 232

Parallel procedures 65, 237
DECOS 197
PIP 197
Posterior probabilities 231
Prior probabilities 53, 231
Probabilities 20, 23, 53, 120, 170, 178, 229

Probability distribution function (pdf) 120
Production rules 12, 170, 180
PROLOG 222
PROSPECTOR 190
PUFF 178

R1 197
Recursion 184, 226
ROSIE 198

SACON 197
SAGE 198
SECHS 197
Sequential procedures 65, 237
Similarity 122
SOPHIE 197
Standard Deviation (sd) 121, 234
SYNCHEM 197
Su/x 197

TEIRESIAS 176

Unit variance 121

Variance 77, 121, 234
VM 198

n/expert-index-1

A message from the publisher

Sigma Technical Press is a rapidly expanding British publisher. We work closely in conjunction with John Wiley & Sons Ltd. who provide excellent marketing and distribution facilities.

Would you like to join the winning team that published these highly successful books? Specifically, **could you successfully write a book that would be of interest to the new, mass computer market?**

Our most successful books are linked to particular computers, and we intend to pursue this policy. We see an immense market for books relating to such machines as:

DRAGON
THE BBC COMPUTER
APPLE
TANDY
SINCLAIR
OSBORNE
ATARI
IBM PC
SIRIUS
NEWBRAIN
COMMODORE

and many others

If you think you can write a book around one of these or any other popular computer — or on more general themes — we would like to hear from you.

Please write to:

Graham Beech
Sigma Technical Press
5 Alton Road,
Wilmslow,
Cheshire, SK9 5DY,
United Kingdom.

or, telephone 0625-531035