INDUCTION BY A HILBERT HYPERCUBE REPRESENTATION

DAVID BALL

Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

August 1991

The University of Aston in Birmingham

INDUCTION BY A HILBERT HYPERCUBE REPRESENTATION

DAVID BALL

Doctor of Philosophy

1991

Summary

This thesis describes a novel connectionist machine utilizing induction by a Hilbert hypercube representation. This representation offers a number of distinct advantages which are described. We construct a theoretical and practical learning machine which lies in an area of overlap between three disciplines - neural nets, machine learning and knowledge acquisition - hence it is refered to as a "coalesced" machine. To this unifying aspect is added the various advantages of its orthogonal lattice structure as against less structured nets. We discuss the case for such a fundamental and low level empirical learning tool and the assumptions behind the machine are clearly outlined.

Our theory of an orthogonal lattice structure the Hilbert hypercube of an n-dimensional space using a complemented distributed lattice as a basis for supervised learning is derived from first principles on clearly laid out scientific principles. The resulting "subhypercube theory" was implemented in a development machine which was then used to test the theoretical predictions again under strict scientific guidelines. The scope, advantages and limitations of this machine were tested in a series of experiments.

Novel and seminal properties of the machine include: the "metrical", deterministic and global nature of its search; complete convergence invariably producing minimum polynomial solutions for both disjuncts and conjuncts even with moderate levels of noise present; a learning engine which is mathematically analysable in depth based upon the "complexity range" of the function concerned; a strong bias towards the simplest possible globally (rather than locally) derived "balanced" explanation of the data; the ability to cope with variables in the network; and new ways of reducing the exponential explosion. Performance issues were addressed and comparative studies with other learning machines indicates that our novel approach has definite value and should be further researched.

Keywords: Learning, connectionist, Hilbert hypercube, subhypercube, minimum polynomials.

To my parents.

ACKNOWLEDGEMENTS.

I would like to express my gratitude to the following people:

To my supervisor, Dr. A.J. Harget: for his interest, encouragement and shrewd advice.

To my wife, Ann for her patience, understanding and support throughout this work.

To my sister, Ros for reading through many early drafts of the thesis.

To my Aunty, Dorothea for lodgings in Birmingham.

To my parents for their general helpfulness.

To Gino Bellavia for the no small task of running the computer laboratories smoothly.

To my fellow research students:

Jacqueline Archibald, Simon Beesley, Nadia Benjeddou, Hugh Dorans, Brian Drabble, Richard Gatward, Ian Hardy, Mansoor Hassan, Ian Johnson, Kevin Lawson, Wathiq Mansoor, Hyacynth Nwana, Mike Parkes, Abdellah Salhi, Hanifa Shah, Sarbjit Singh, Neil Simpkins, Robert Thomson and Irene Woon,

for their lively conversation, friendship and moral support.

To Dr. Brian Gay for allowing me to attend various conferences, support in publishing and for providing the department with excellent computing facilities.

To Dr. Brian Smith for several interesting medical discussions on urology as an application area.

Finally I acknowledge the financial support of **SERC**.

PREFACE

Of what value is this book?

Every author would like the reader to sit back and entertain the author's perspective on things for a while. That is the joy of books. In some research work the value is mainly in the experiments performed and results obtained. In other research work the value is mainly in the mathematical proofs and rigour employed. In the case of this thesis, the author asks the reader to consider the following observation.

"Imagination is more important than knowledge."

Albert Einstein

List of Contents

List of Figures

List of Tables

Plates

Constructed Hypercube Models.

Plate 1: H^1, Line; H^2, Square.

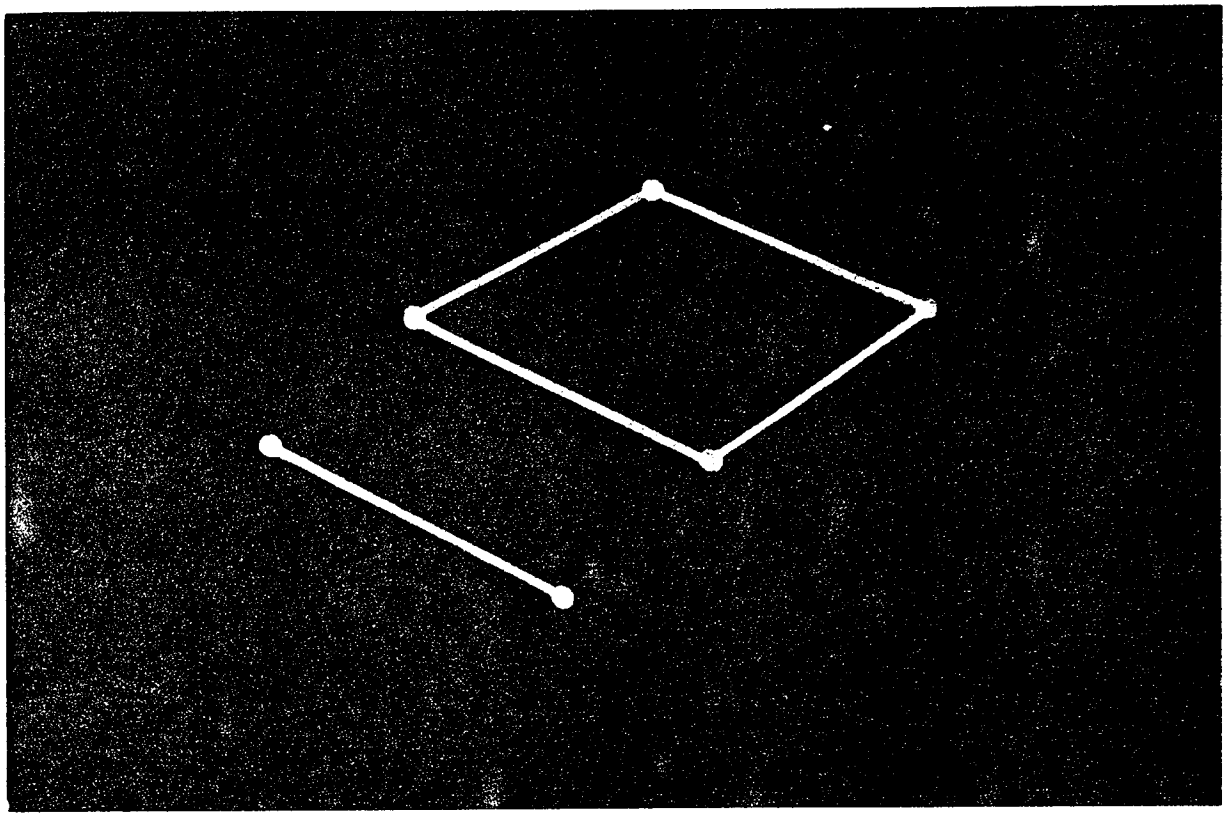


Plate 2: H^3, Cube.

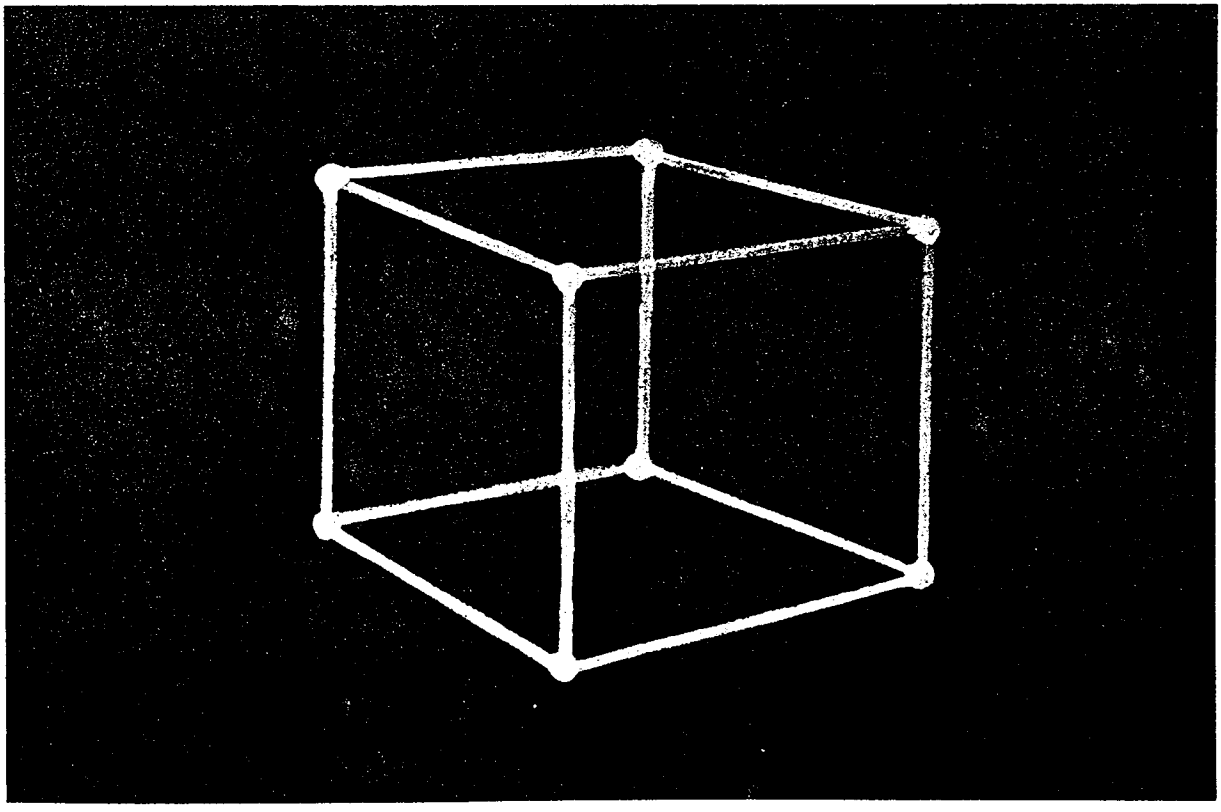


Plate 3: H^4, Necker's Cube.

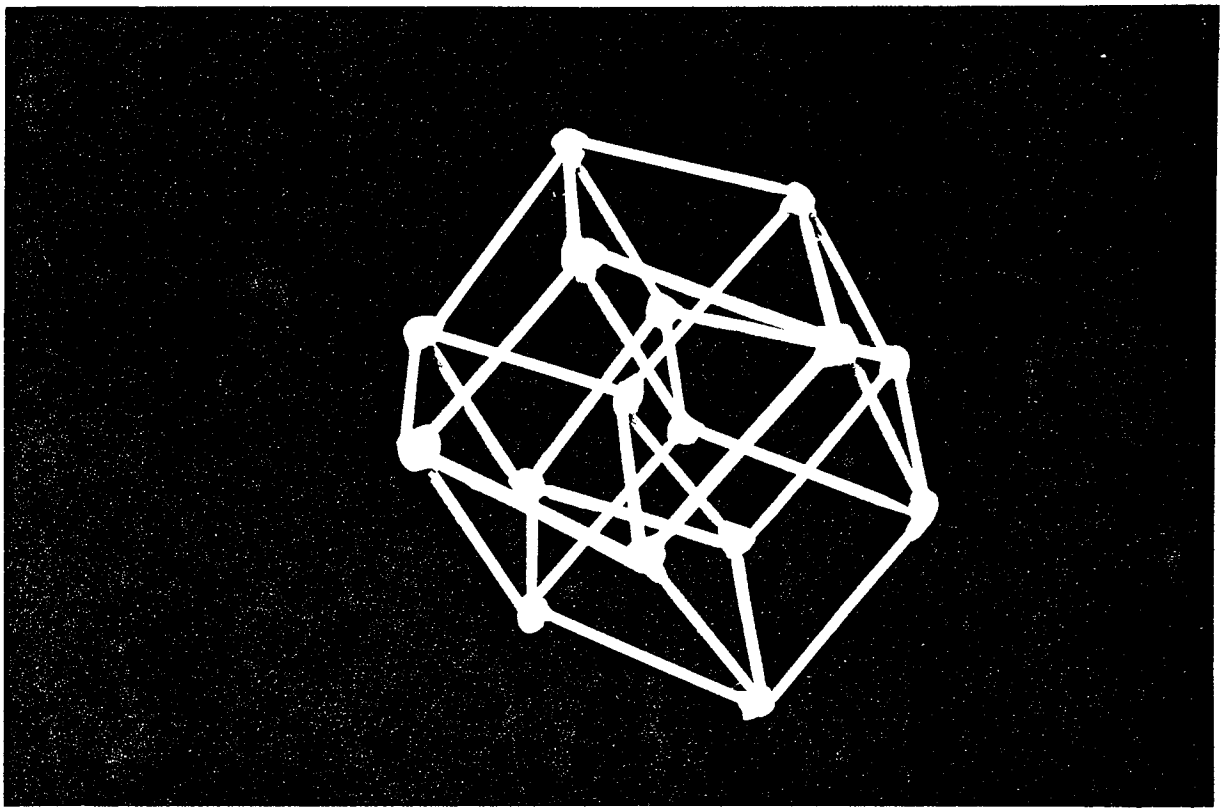


Plate 4: H^4, The Tesseract.

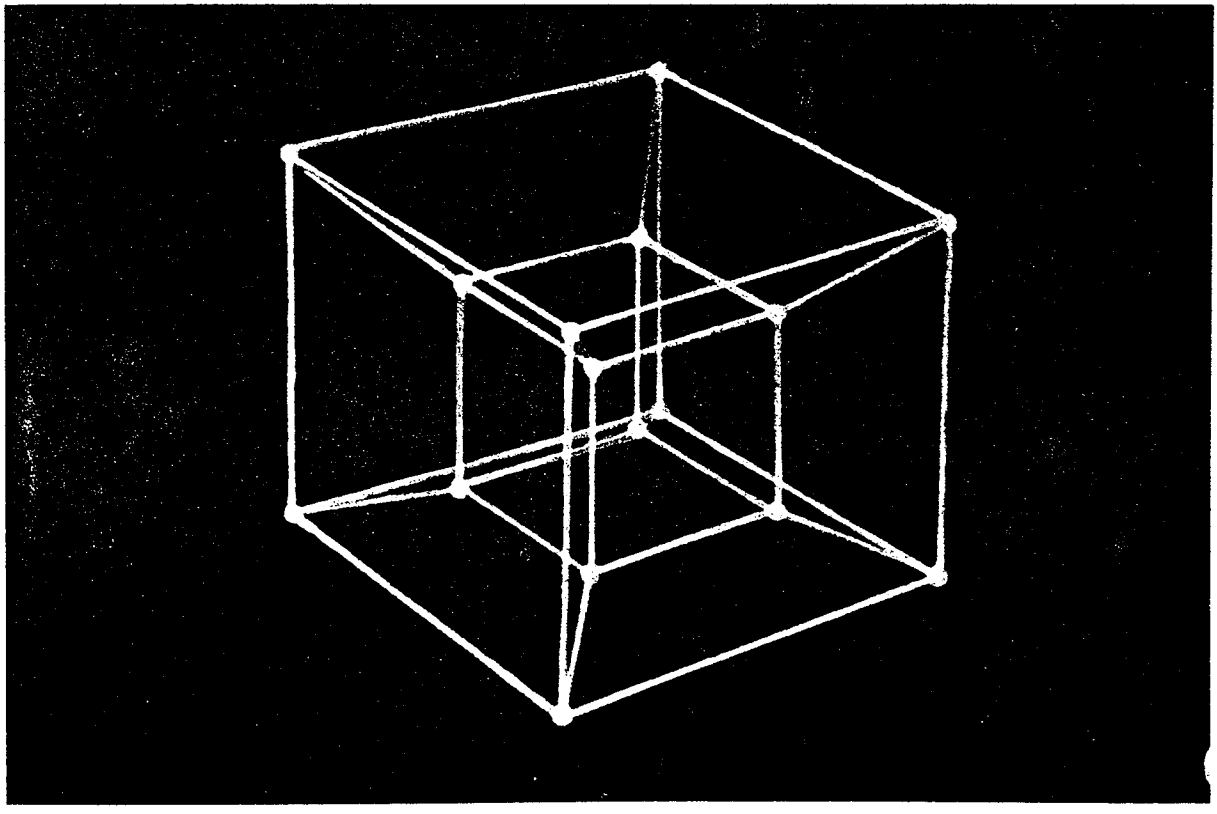


Plate 5: H^5, The Hypertesseract (front view).

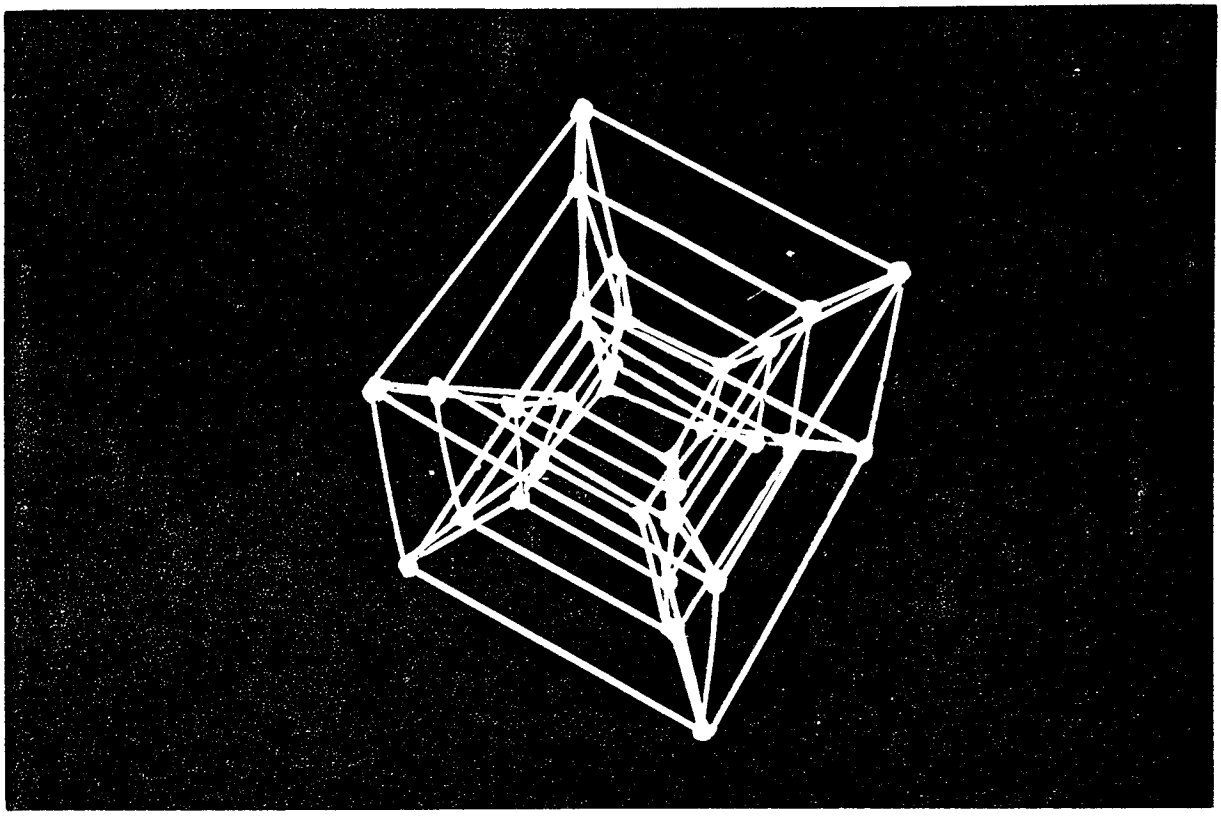
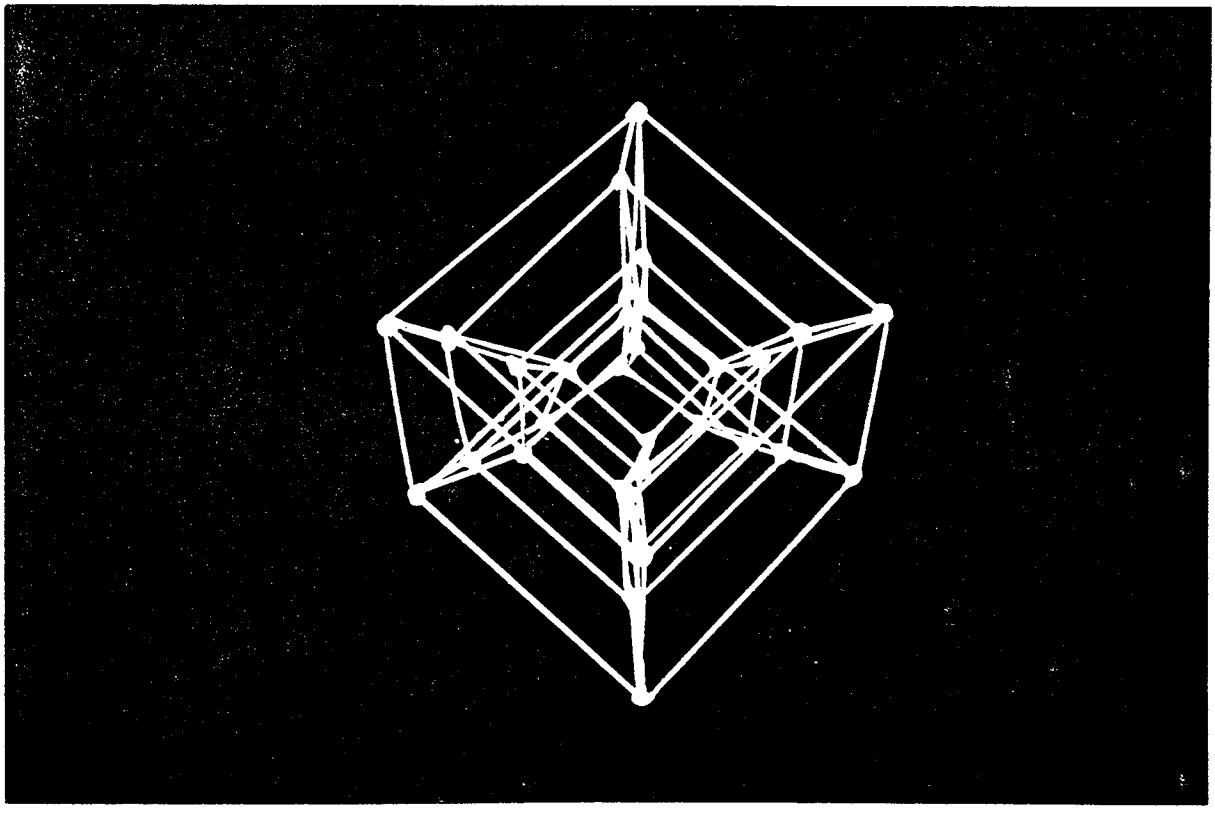


Plate 6: H^5, The Hypertesseract (side view).



CHAPTER 1

Introduction

1.1 A BRIEF HISTORY OF LEARNING.

It has been something of a surprise to find that computers have proved to be excellent at things that we find difficult, and unacceptably poor at things that we do effortlessly. The subject of Artificial Intelligence or AI has attempted to redress this latter deficiency. In attempting to make progress with the conceptually difficult subject of AI, one aspect of intelligence is intuitively felt by most researchers to be crucial, namely learning. The great variety of extant learning machines have been developed within the historical context of a war between antagonistic tendencies, called symbolic and non-symbolic or connectionist.

1.1.1 The First Connectionist Period.

In the 1940's biological nerve cells or "neurons" were seen as the apotheosis for a series of simple abstractions or elementary "neural networks" by interconnecting simulated biological neural elements. In 1949 a psychologist, Hebb produced a classic book called "The Organization of Behaviour" (Hebb, 1949).

The basic idea was that networks of neurons learn by constructing internal representations of concepts, that is, that "*connections between mechanisms* rather than the *connections between symbols* will ultimately explain psychology". Further Hebb held that when two cells involving a single synapse are active together then learning takes place. These two assumptions are still the rationale behind current neural networks.

The early 1950's research discovered the importance of learning. Learning is ultimately necessary, since just as we need to do much that is not already built-in, so there will never be enough time to build-in all that an AI machine will ultimately need to know.

1.1.2 The First Symbolic Period.

The latter half of the 1950's saw the rise of the serial processing machine based on the Von Neumann architecture via the processing of symbolic expressions. Thus rapid progress with regard to new and difficult problems using the counter of the Hebb theory, that is of *connections among symbols* rather than Hebb's *connections among mechanisms*, and the counter concept of *programming* rather than *learning*, meant the demise of the earlier neural network machines.

1.1.3 The Second Connectionist Period.

However in 1962, Rosenblatt's book "Principles of Neurodynamics" revived an interest in Connectionism (Rosenblatt, 1962). Rosenblatt defined the simplest possible learning machine which he called a "Perceptron" and proved that a Perceptron could, in principle, learn anything that it was possible to program it to do. (Actually Rosenblatt was mathematically unhousetrained! Papert did the the impressive mathematics in the book).

Unfortunately, in practise, Perceptrons were found to be unable to learn in all circumstances. The paucity of further connectionist progress and the advent of Minsky and Papert's critical appraisal entitled "Perceptrons" combined once again to halt the connectionist approach (Minsky and Papert, 1969). Minsky later summarised the nub of the problem as "learning is a non-problem the real problem is representation" (McCorduck, 1979).

1.1.4 The Second Symbolic Period.

In the late 1960's and throughout the 1970's the resurrected symbolic paradigm continued to make progress and saw the emergence of the central concept of the representation of knowledge. Problem-solving, control and hence reasoning only learning machines, such as Newell and Simon's "General Problem Solver", were quickly seen as limited although they became the precursor to modern planning systems (Newell & Simon, 1963). Selfridge had produced an entertaining machine called "Pandemonium" in which there was a symbolic attempt to rediscover the connectionist idea of intelligence resulting from the workings of a large number of simpleton elements (Selfridge, 1959).

Lindsay *et al.* with a machine called "Dendral" then put these two insights together to produce a new sort of machine called an expert system, which was designed to be "expert" at problem solving in some tiny domain, in the sense of being comparable in performance to a human expert *within that field* and which divided the reasoning or "inference engine" from the representational knowledge it used in its "knowledge base" (Lindsay *et al.*, 1980).

This implied a new insight into what constitutes intelligence and comprised the notion that large amounts of domain specific knowledge was of greater importance than a large number of individually clever reasoning processes. This advance, as Randall Davis then showed with his system called "TEIRESIAS", implied that a new kind of symbolic learning process, called a Knowledge Acquisition system, or KA system, is required, to produce the knowledge base (Davis, 1980). That is, a machine is, required to fill the knowledge base by extracting that knowledge, if possible semi-automatically or ideally automatically from a human expert, in the field of the domain.

Unfortunately, KA became a "bottleneck" for expert systems since experts were found to be very poor at elucidating their knowledge as required by, pragmatic, analysis-only, hand-built, non-automatic KA systems. Secondly, the semi-automatic inductive KA systems, based on the fact that experts are much better at giving examples of their knowledge, were and still are at an elementary stage of development.

However a third approach to learning had begun to emerge in the early 1970's as typified by Winston's "Learning Structural Descriptions from Examples" (Winston, 1975) and was a symbolic approach to the study of learning itself, known as Machine Learning or ML, with learning by example being, both then and later, its most studied form.

1.1.5 Knowledge Representation.

We seem to possess a connectionist neural network capable of both low-level non-symbolic processing (e.g. vision), and high-level symbolic processing (e.g. thinking). For example, we can imagine a black cat in mid air a short distance in front of us. We can then enlarge it, shrink it, rotate it, translate it, change its colour, distort it, view it from a different perspective, even talk about it, imagine that we can hear it, touch it, smell it, etc. This appears to imply an incredible, unifying, transferable, knowledge representational capability.

As the 1970's progressed, work on the three approaches to learning (Connectionist, KA and ML) saw reduced progress and the big explosion of research work (of the order of one million papers to date) took place in a multitude of knowledge representational schemes across the whole range of the fields of AI. Examples of these representational systems varied from large grained structures such as blackboards to small grained, modular structures such as production systems, as for example in rule based systems for inclusion in an expert system knowledge base, as in: scripts, schemas, mops, frames, semantic nets, conceptual dependency, plans, various types of natural language parsers, various logics such as fuzzy logic and non-monotonic logic, various image processing representations etc. Thus the importance of powerful, flexible knowledge representational systems and processes to manipulate them had been well learnt and was beginning to bear fruit, particularly in the field of expert systems. The 1980's saw the continuing confluence of the symbolic and representational fashion in AI and significant rejuvenated progress on the theory of ML and a widening variety of practical approaches to expert systems and KA systems.

1.1.6 The Third Connectionist Period.

However as the decade progressed, there has been renewed interest in the connectionist paradigm world-wide. One of the prime motivating forces has been the gradual realisation that even multiple paradigm approaches (discounting the disadvantages of their

dimly perceived juxtaposition problems and consequences), such as rules within frames, produced little of commercial use in the "real" world apart from the tentative hold of the emerging "expert systems", and also new and more powerful computers had led to more ambitious and novel architectures by the diehards such as Kohonen and Aleksander and newcomers such as Rumelhart, McLelland, Hinton and Grossberg.

So as the 1990's begin, this rediscovered connectionism is spreading not just across most of the fields of AI but also inspiring diverse subjects such as psychology, neurophysiology, neurobiology, physics and pattern recognition in mathematics and seems set to become the next trend. Yet, many modern connectionist "discoveries" are to be found in the pages of Rosenblatt's book, and earlier works. Similarly, to slowly forget the practical successes of symbolic AI such as expert systems and its required KA system and the theoretical foundation work in ML, and to unlearn the crucial lesson of the importance of knowledge representation, in the renewed excitement for neural networks seems to be unthinkable and doomed to be wastefully rediscovered later, in yet another round of the symbolic connectionist tussle. Might there therefore be another way?

1.2 LEARNING ENGINE CRITERIA.

Of particular interest in evaluating learning engines are the following criteria; "input felicity" (how easy is it to construct the database for input to the learning engine), "learning speed" (how fast does it learn, is it iterative or one-shot in operation, and is convergence assured or only obtained by trial and error), "noise immunity" (to what extent will noise invalidate the results), and accessibility" (can the machine explain where it has got to, what it is doing, and why and how it is doing what it is doing).

In addition to these criteria a learning engine is required to be application independent, able to operate in the symbolic paradigm and/or in the connectionist mode, able to work with information at a high level such as knowledge, and/or at a low level using objective real world data, and should be fully automated, that is, able to continue without the occasional intervention by an expert user, who may otherwise be required, for example to facilitate the input data. Most important of all is the need for an exceptionally rich and unifying representation which is able to transfer information between levels, but no learning engine, to date, satisfies this requirement. Previous learning engines can be divided into three groups, described below, with reference to these criteria.

1.2.1 Connectionist Machines.

Connectionist machines are application dependent, non-symbolic, low level working, at best, on objective real world data, and are automated. Input felicity is not usually

too much of a problem since a "toy" ready made database may be used or constructed or even, real world data used directly. However learning speed is the accepted big problem area for connectionist machines on two counts.

Firstly the process itself is heavily iterative, typically taking hours or days for simple real world problems and possibly very much longer. The reasonableness of and need for this state of affairs is questionable. For example biological neurons work, by comparison, very slowly, with firing rates of the order of 100ms. Yet we appear to be capable of single example, incremental learning, rather than the thousands to millions of iterations required by batch-mode connectionist machines. For example, witness the single example learning the reader is at present undergoing.

Secondly the algorithm may not converge, or do so unacceptably slowly, implying a lengthy period of possibly weeks of parameter juggling. This unpredictable, trial and error, hand-build approach is also questionable. Noise immunity is a big strength of connectionist networks, but accessibility is nonexistent. If the machine "gets stuck" it can be almost impossible to figure out what is going on, and it certainly cannot explain itself. However connectionist networks, *per se,* have numerous other advantages and a strong interdisciplinary appeal.

1.2.2 Knowledge Acquisition machines.

Knowledge Acquisition systems are application independent, symbolic, high level by acquiring knowledge from human experts, but are only, at best, semi-automated. Input felicity is a problem area for expert systems, implying as it does an iterative cross-questioning of the human expert in the field, and unfortunately experts are often weak at elucidating their knowledge. Knowledge Acquisition systems are designed to address this problem and hence inherit the difficulty.

Learning speeds are very acceptable, especially considering that expert systems are nowadays applied to real world problems, although semi-automated inductive Knowledge Acquisition systems can still only handle elementary real world problems. Noise immunity is variable depending upon the system. Accessibility is the big strength of Knowledge Acquisition systems, in that the machine itself can explain its actions, as can the target expert system.

1.2.3 Machine Learning Machines.

Machine Learning systems are application dependent, symbolic, high-level typically working on data from a "toy" world, and are usually semi-automated. Input felicity tends to

be straightforward since a ready-made database is often used, and learning speeds are comparatively very fast. Against these strengths, it is to be remembered that despite considerable theoretical work, Machine Learning is still at an early stage of development, and real world problems are therefore a difficulty. Noise immunity is a big problem area and accessibility is weak, although possible with an effort on the part of the user, since the process handles symbolic information.

1.3 TOWARDS A COALESCED MACHINE.

To summarise the above discussion, the relative merits of the three learning paradigms with respect to the stated evaluation criteria are given in Table 1.1.

		Evaluation Criteria			
		Input Felicity	**Learning Speed**	**Noise Immunity**	**Accessibility**
Machine	**Connectionist**	easy	unacceptably slow	very good	very poor
	Knowledge Acquisition	very difficult	fast	variable	very good
	Machine Learning	easy	fast	very poor	poor

Table 1.1 Evaluation of different learning machines

Each learning paradigm appears, at present, to be inescapably stuck with its strengths and weaknesses, since they tend to delimit and are the essence of the paradigm. Yet we saw in the above "learning engine criteria" (section 1.2) that the requirements for learning engines are diverse, certainly covering the strengths of each of the types so far developed. Further, the AI lessons of the last two decades, on the crucial importance of attention to Knowledge Representation, must not be forgotten. Thus interest shifts to the possibility of an amalgam of the three machines in a "coalesced machine" - hopefully attempting to retain the strengths of each whilst eliminating the weaknesses of all three.

Let us not labour the point. Taking our cue from the importance of representation in AI we are **herein** interested in developing what we see as a *necessary and unifying representation* in order to accommodate the three learning paradigms and to facilitate the transfer of information between the connectionist and symbolic parts of that machine.

In summary AI has thus oscillated in fashion between the two major paradigms of symbolic and connectionist. We are interested in the middle ground, rather than this dichotomous prospect; that is, in the problem of how machines of these two types may interface with one another within the context of learning and what this may imply for either or both types of machine and further, in the coalescing of the three major approaches to learning. The structure of the thesis therefore follows the outline below.

1.4 THE STRUCTURE OF THE THESIS.

In Chapter 2 the first three parts outline the approaches taken by other researchers to the problem of learning in regard to the three main paradigms of Machine Learning, Knowledge Acquisition and Connectionism. We then examine the strengths, weaknesses and dangers of existing and possible integrated machines. Finally interesting implementations of Machine Learning, Knowledge Acquisition and Connectionist paradigms are examined in greater detail with a view to later coalescing them into one machine in order to attempt to overcome some of the limitations of each, whilst retaining the advantages of all three.

Chapter 3 gives the theoretical details, foundations and aims of the design and implementation of the above coalesced learning engine able to address the interface problem between the symbolic and non-symbolic approaches via a unifying representation.

Chapter 4 describes the development of the new machine and the details of and manner in which experiments to determine its behaviour were investigated in order to delimit its scope and scale. The results obtained using this novel machine are then discussed and compared and contrasted with contemporary learning engines.

Finally, in Chapter 5 we offer our major conclusions made on the basis of this study as to to the relative merits of the coalesced machine.

CHAPTER 2

Learning Machines: Theory and Practice.

2.1 AN OVERVIEW OF LEARNING.

Michalski *et al.* (1983) open the preface of the first of their excellent books on Machine Learning with the words: "The ability to learn is one of the most fundamental attributes of intelligent behavior." This certainly seems to be inescapably true. For example, if an animal is either inherently incapable of learning anything or has very minimal learning capabilities then we tend to regard it as unintelligent, irrespective of whatever other capabilities it has.

Carbonell *et al.* (1983) take the study and computer modelling of learning processes in their multiple manifestations as constituting the subject matter of "Machine Learning." Carbonell further observes that learning is a multifaceted phenomena such that the processes of learning include acquisition of new declarative knowledge, the development of motor and cognitive skills by instruction or practice, the organization of new knowledge into general representations and the discovery of new facts and theories through observation and experiments.

2.1.1 Aims and Benefits of Learning.

Carbonell then states three objectives of machine learning. Firstly, theoretical analysis explores the space of possible learning methods and algorithms independently of the application domain. Secondly, task oriented studies which aim to improve performance in set tasks, corresponding to an engineering or applied learning approach. Thirdly, cognitive simulation implying the investigation and computer simulation of human learning processes. Thus the three major approaches form the follcwing goals:

1. Intelligence Science.
2. Knowledge Engineering.
3. Cognitive Modelling.

Carbonell further propounds that the theoretical study of learning may reveal general and invariant principles of intelligent behaviour that apply across many different domains. For example, concerns such as the exploration of alternative learning mechanisms, the discovery of various induction algorithms, the acquisition of concepts, the scope and limitations of methods, the information that must be available to the learner and robustness in the presence of noise. In short an "intelligence science" possibly entailing three major and

interconnected aspects, namely: learning, the use of language and neural. As stated in the introduction, in section 1.1.1, one of the main advantages of machine learning is that we may be able to reduce the amount of programming required. For example, Carbonell *et al.* (1983) argue that machine learning research strives to open the possibility of instructing computers in new ways and thereby promises to ease the burden of hand-programming the growing volumes of increasingly complex information into the computers of tomorrow.

A practical example would involve expert systems which despite their current success often require many man-years to construct, perfect and maintain. The bulk of this work goes into developing and debugging extensive domain-specific knowledge bases. A better understanding of learning, therefore, may allow the automatic construction of the knowledge base. This constitutes the "knowledge engineering" approach.

The theory and modelling of learning is also useful to all intelligence fields: cognitive science, artificial intelligence, pattern recognition, psychology and education. The spin-offs of this "cognitive modelling" approach are inspirational. For example, by modelling human learning and teaching we may improve the design of teaching methods and create the possibility of intelligent automated tutoring systems. Hence machine learning has a strong interdisciplinary appeal. With the present rise of connectionism, noted in the introduction, physicists and neuroscientists are also making substantial contributions to the field. Therefore it is first necessary to dissect this huge field of learning in order to delimit aspects of interest to this present work.

2.1.2 Research Directions in Machine Learning.

There are many approaches to learning, both from the viewpoint of building models of human learning and from the perspective of understanding how machines might be endowed with the ability to learn. Examples include work on genetic learning algorithms (Holland, 1975; Oosthuizen, 1987b), connectionist models of learning (Hinton *et al.*, 1984), knowledge acquisition for expert systems (Kahn, 1986), and grammar acquisition (Mitchell et al., 1986).

2.1.2.1 A Taxonomy of Learning.

The subject of "Machine Learning" (not to be confused with machine learning, the generic term) has the most thorough theoretical classification of the various branches, categories, theories and types of learning. It is more cohesive to follow a somewhat ***historical*** perspective. In the "bible" of Machine Learning, Michalski *et al.* (1983) consider that the major research directions are as stated below.

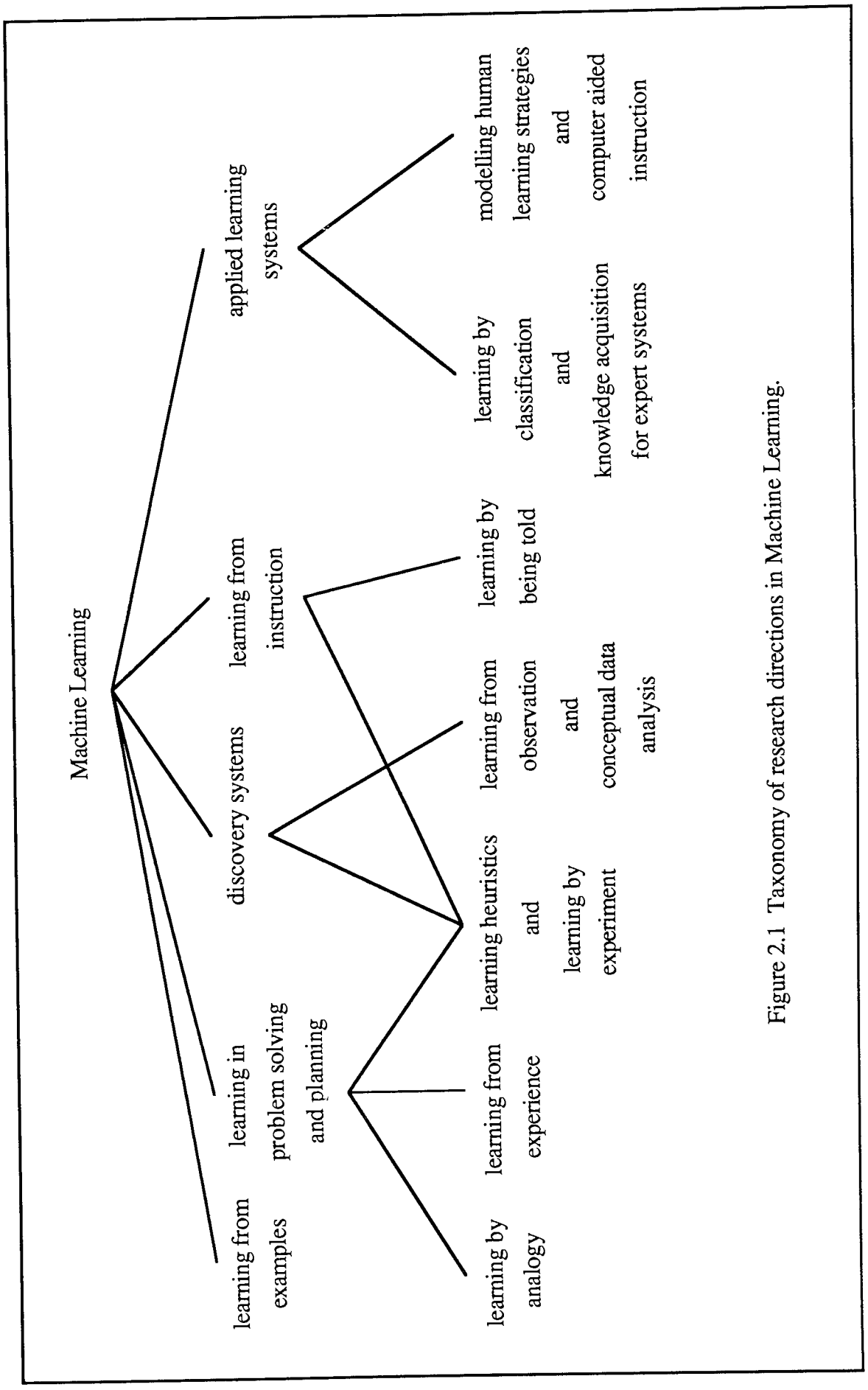


Figure 2.1 Taxonomy of research directions in Machine Learning.

1 Learning from examples.
2 Modelling human learning strategies.
3. Knowledge acquisition for expert systems.
4 Learning heuristics.
5. Learning from instruction.
6. Learning by analogy.
7. Discovery systems.
8. Conceptual data analysis.

The interrelationships between these approaches are given in Figure 2.1.

2.1.2.2 Automation Level.

It is possible to classify the various machine learning systems along a number of dimensions, such as, classification on the basis of the representation used, classification in terms of the application domain or classification by the learning strategy used. Consideration of the level of automation clearly delineates the various research directions in an understandable manner. Figure 2.2 gives a pseudo-graph to illustrate this concept as proposed by Carbonell *et al.* (1983). Each type of learning is seen as having two parameters; the amount of effort afforded by the teacher and amount of effort afforded by the learner. Various learning approaches are then plotted on the graph accordingly.

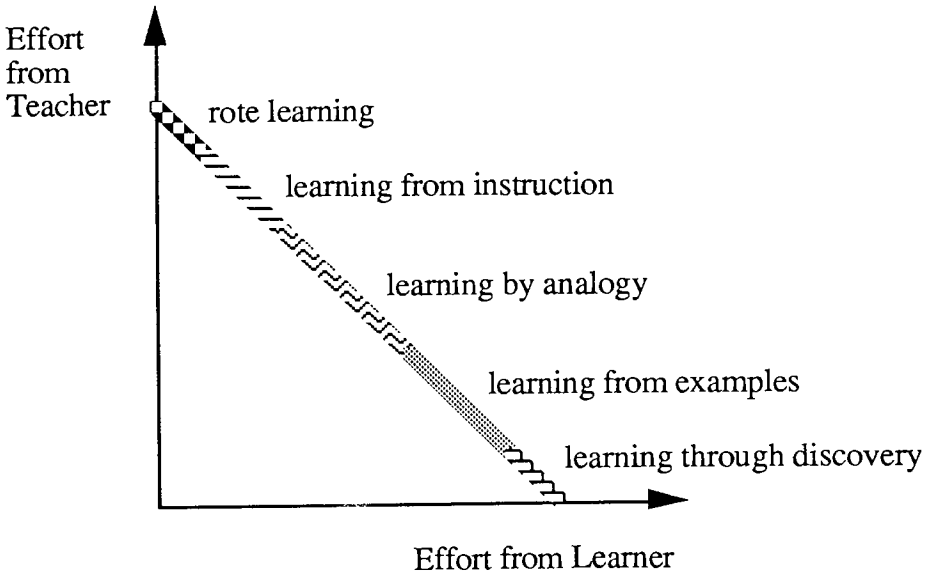


Figure 2.2 Effort from Teacher against Effort from Learner

Rote learning occurs when the learner is **not** required to contribute any effort and therefore all analysis, structuring etc. is done by the teacher. Conventional programming is an example of rote learning.

From this extreme point on the graph, by reducing the contribution from the teacher and increasing the contribution from the learner, we progress through: learning from instruction, learning by analogy and learning from examples, to learning through discovery. The final extreme point corresponds to **no** prior analysis or help from the teacher, and all the work must then be done by the learner. The word "teacher" is to be taken liberally, for example, meaning the physical environment which contributes the input to the machine.

In learning from instruction, an external entity organises the information to aid learning. That is, the information is part pre-digested and then the learner can apply this instruction to novel situations. Two types of learning from instruction are advice taking systems (Mostow, D.J., 1983; Hass & Hendrix, 1983) and learning apprentice systems (Mitchell *et al.*, 1986). Learning apprentice systems show most promise since experts are good at stating the appropriate action in some particular situation in order to get to the solution. Such systems are heuristic and take a problem reduction approach to heuristic learning, and involve credit assignment by assuming the expert's moves are positive instances and alternatives are negative.

In learning by analogy the past, as given by the teacher, becomes a clue to new and possibly more complex concepts *deduced* by the learner (Carbonell, 1983, 1986a). Learning by example implies a system with an external teacher otherwise it is learning by experience. Such systems utilise positive examples and exclude negative examples to *induce* positive concept generalisations (Winston, 1975; Dietterich & Michalski, 1983; Mitchell *et al.*, 1983; Quinlan, 1983).

Learning through discovery lacks an explicit goal and an active external teacher and may involve a range of interactions with the environment from active experimentation to passive observation (Michalski, 1983; Lenat, 1983; Langley *et al.*, 1983; Michalski & Stepp, 1983). We will briefly consider some examples of these types of machine.

2.1.2.3 Learning from Instruction.

Advice taking systems have the objective of transforming declarative advice into executeable procedures, by successive heuristic transformation of the advice into specific operational terms. Much prior knowledge is required in terms of knowledge of the advice representation, the actions that are available for implementing the advice and the constraints of the domain.

The idea behind learning apprentice systems is that the machine begins by using means-ends analysis or some other general purpose problem solver in order to obtain a faster set of operators as specialised new domain knowledge. The system observes how the

teacher works as an expert and converts declarative knowledge into its equivalent procedural form. The expert may coach the system via a set of suggested preferences. The machine, from a self analysis of its past experiences, then learns to avoid mistakes and inefficiencies. Environmental feedback provides quality control and operator refinement.

2.1.2.4 Learning by Analogy.

A major contribution to learning by analogy was made by Carbonell (1983, 1986a). A basic technique is Derivational Analogy. Derivational Analogy works by a transformation process illustrated in Figure 2.3.

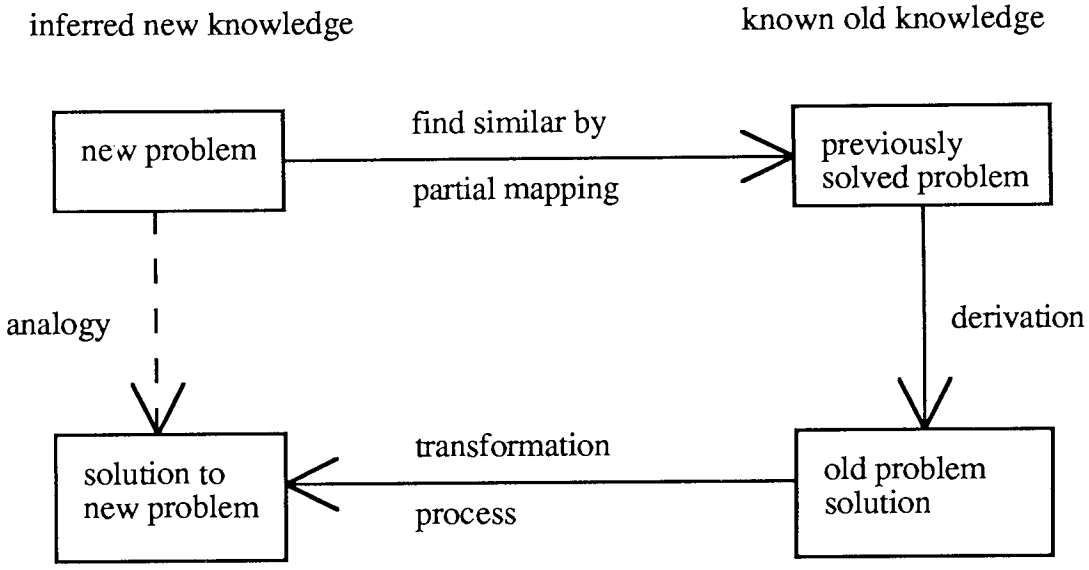


Figure 2.3 Derivational Analogy

A similar problem is extracted from known problems, which have previously been solved, by a process which forms a match between the known old knowledge and the required new knowledge. Clearly, the closer or more similar these two are, the easier it becomes to search for an analogy. The system recalls the previous solution to the similar old problem and transforms the recalled solution in order to satisfy the constraints imposed by the new problem.

A critical point is that an analogy can only be produced if it is *noticed.* Hence the search time is important. For example, the simple geometrical analogy in Figure 2.4, employed by Anderson & Kline (1979) takes 12 minutes to run (Carbonell, 1986b). Derivational analogy can be combined with explanation-based learning. The more **n** problems that are solved the greater is the likelihood of solving problem **n + 1**, because more solutions exist. Unfortunately, useful analogies are rare.

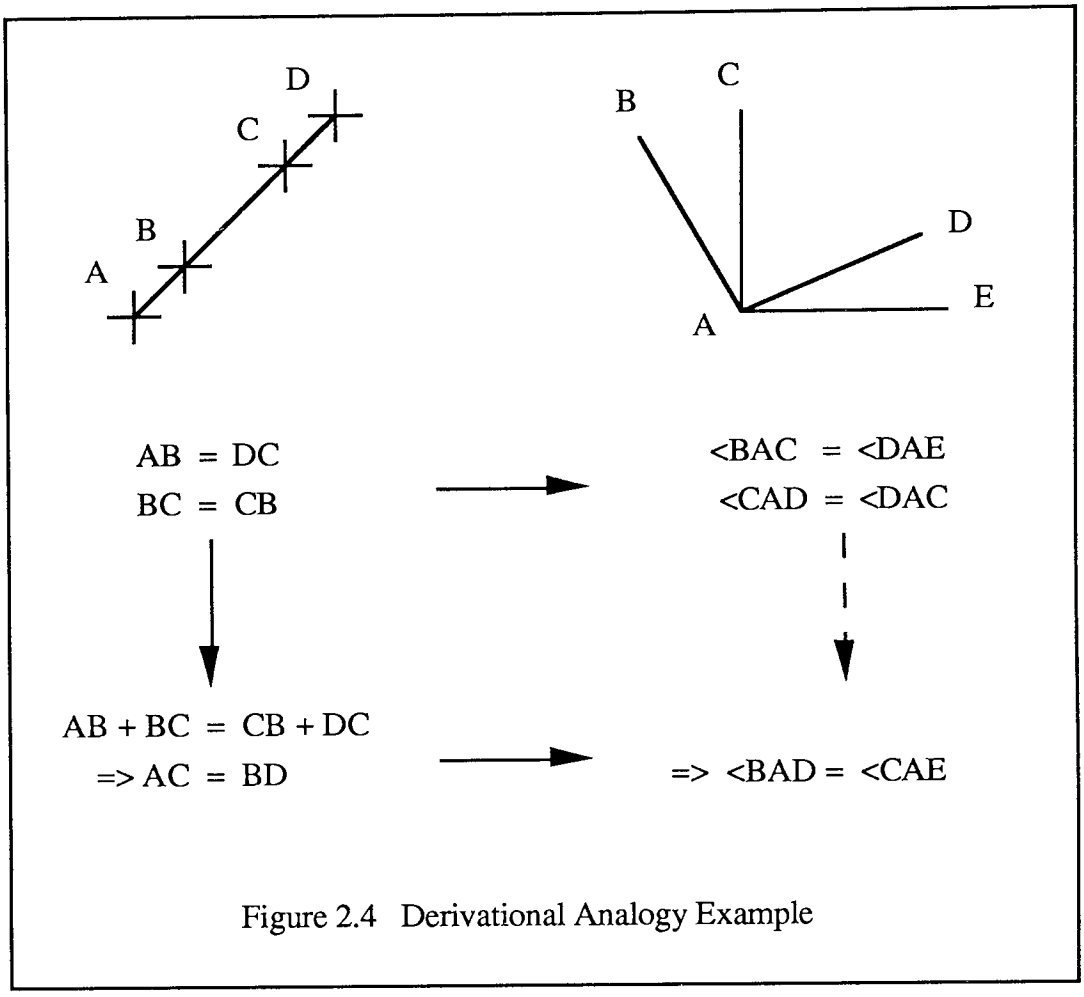


Figure 2.4 Derivational Analogy Example

2.1.2.5 Learning by Example.

Clearly all learning can be regarded as "by example" and its basic methods form part of more complex learning systems. Learning from examples is the most widely studied problem in Machine Learning. For these reasons "learning by example" has become one of those horrid terms meaning all things to all men. What an author means by the term may therefore be unclear.

At one extreme it means practically any form of learning since it is difficult to find any form of machine learning which does not in some sense use examples. The "by example" part of "learning by example" then becomes superfluous. At the other extreme, "learning by example" means a particular low level sub-unit of a more general learning system, as we will now consider.

The method involves a set of given positive and negative instances of one or more concepts. For example Winston (1975) showed the importance of near misses in constraining the resulting concept description. The goal of the system is to generate a

description in the form of some representation such as rules that covers all positive instances, the "completeness condition", and no negative instances, the "consistency condition". The part played by the teacher is typically that of a tutor who clusters the objects concerned in order to ease the task of the learner. The tutor, therefore, often does a lot of the work. The representation used may be simplified by the data and concepts having the same format.

At its simplest level the "learning by example" task is illustrated in Figure 2.5 In general, an incomplete "training set" from some "universe" of possible instances is used and comprises a set of positive and a set of negative examples. The positive examples are used to generalise or increase the scope of the concept space and the negative examples are used to discriminate or bound the scope.

The general conclusion, typically as a concept, is then tested against a possibly incomplete "test set" from the universe. The test set and training set are usually disjoint parts of the universe, as below.

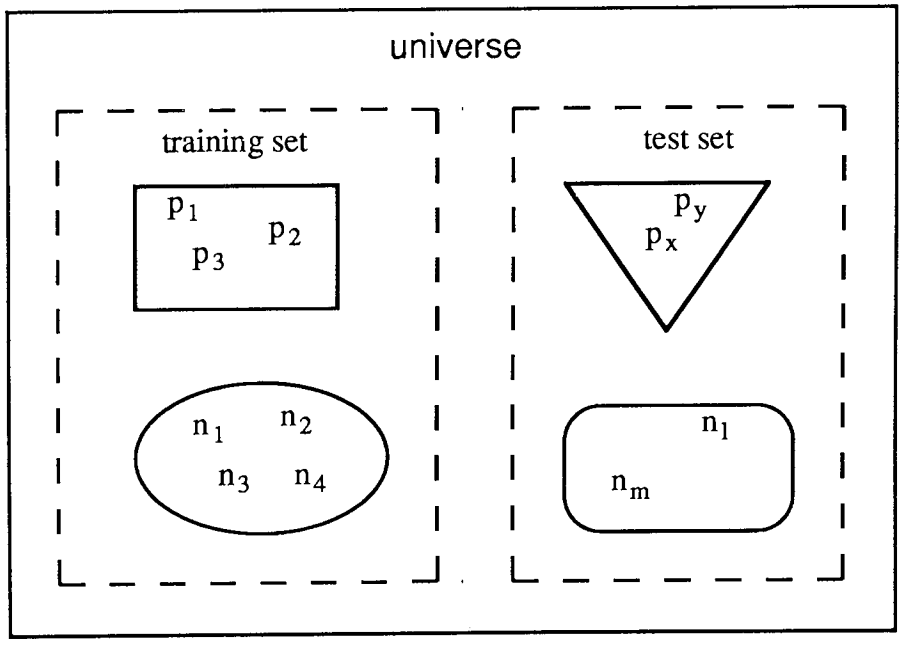


Figure 2.5 Learning by Example.

Hence the learn-test process concludes concepts C which include all of the positive examples and none of the negative examples, that is, such that for the general ith example E:

$$E_C \Rightarrow \{ \begin{array}{l} \forall E_{pi},\ E_C \supset E_{pi} \\ \sim\exists E_{ni},\ E_C \supset E_{ni} \end{array}$$

There are many types of such learning engines due to the range of possible parameters (Kodratoff, 1988). The descriptors involved may be finite-valued, or they may form a range or a continuum, or be hierarchical, etc. The method used may be heuristic or involve some lattice of possibilities. The examples may be externally or internally generated. The generalisations may be conjunctive, disjunctive or both. The system may be incremental or one-shot in operation and so forth, as shown in Figure 2.6.

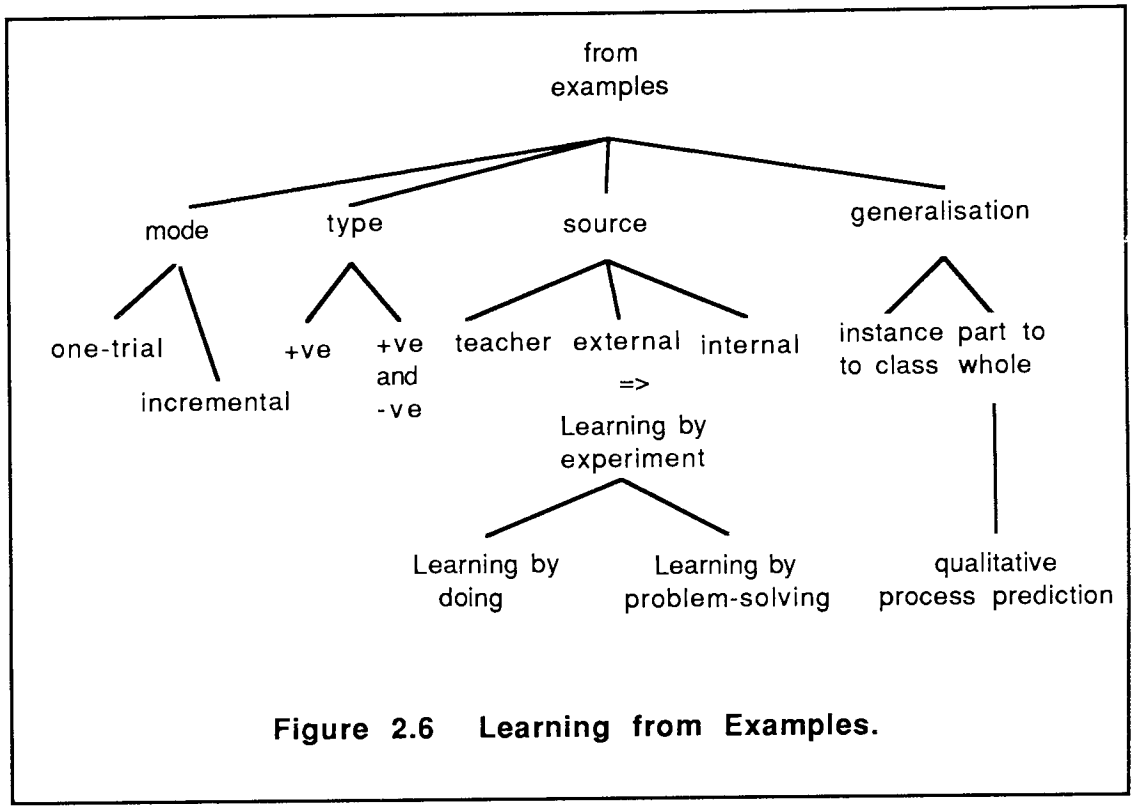


Figure 2.6 Learning from Examples.

What is a concept? If we take it to be an hypothesis about some aspect of the world then these hypotheses can be written in the form of descriptions. A common approach is as follows.

The set of descriptions considered in learning from examples can be ordered according to their generality. This defines a partial ordering through the space of possible hypotheses. The resulting multiple paths through the representation of states in this problem space can be searched via a set of operators, as discussed by Mitchell (1982), for the "correct" concept as a goal.

Clearly one might start the search anywhere in this space but two natural starting points are the most general description and the most specific description. Some algorithms combine both search directions, thereby moving towards more general concepts in some cases and towards more specific concepts in other cases. The most specific description is so called because it contains the most detail i.e. lowest subset.

This defines the dimension of generality, the partial ordering of the hypothesis space being required to direct the search. The fact that the generality ordering is only *partial* is important. Firstly, it defines a *lattice* of possible concept descriptions. Secondly, it implies that some hypotheses are related along the general to specific dimension, while others are not related. For example, A and B may be more specific than C, but not more specific than each other.

Michalski (1983) in his excellent work on the theory of inductive learning considered various representations of generality including the following, in order to create more specific descriptions (or reversely for more general descriptions).

1. Adding conditions.
2. Replacing variables with constants.
3. Removing elements from lists.
4. Climbing the specialisation tree.
5 Replacing terms in isa hierarchies with a lower term.
6. Decreasing the size of intervals.

For example, the adding/deleting conditions implies that reducing the ANDed rule size implies a more general description. In interchanging variables and constants (a common technique), if a certain rule specifies a particular number and this is subsequently replaced by a variable then the first description is more specific than the resulting more general second rule. A number of other theories exist (Vere, 1980; Dietterich & Michalski, 1981; Bunday & Silver, 1982; Kodratoff & Ganascia, 1986; Valiant, 1985; Delgrande, 1988; Haussler, 1988; Gallant, 1990; etc.).

Kodratoff and Ganascia (1986) in an evocative paper considered how to improve the generalisation step. The main points of which are that, firstly, concept discovery implies discovering the variable bindings and secondly, information may be dropped from the formula only with extreme care. They detect variable bindings common to the positive examples (converting constants to variables) and use the dropping the condition rule as discussed in Michalski (1983) only for negative or counterexamples as a source of possible near misses (Winston, 1975). The generalisation is accomplished in two steps. Firstly, detection of structural matchings takes place. This, state Kodratoff and Ganascia (1986), is the difficult and necessary step where the real work is done. Secondly, there is the generalisation phase, proper, which detects common links between variables in all the structurally matching formulas. This is the easy step. The structural matching algorithm is a sequence of two alternative operations which find new generalisation variables (GV) common to all examples so far (links between variables and constants/variables track the variables in each example).

Firstly, constants/variables which are not GV in each example are chosen by a heuristic method in order to produce new GV. However, the actual heuristics required to do this are unspecified in the paper and bearing in mind the emphasis on deduction in this paper it may well be the case that these heuristics are domain dependent. This has all sorts of other implications (as we discuss in more detail later), nevertheless failure to mention this crucial point is a weakness. Briefly mentioned are the possibilities of the user ordering the examples, the user choosing directly for each example and combinatorial search as the present state of implementation.

Secondly, there is a partial matching phase which looks for structural matchings of GV in the subset of examples. These two stages in the algorithm are repeated until there are no more constants.

Nevertheless, as seen above, the big problem area is the difficulties of finding appropriate heuristics and/or how to search the hypothesis space as fast as possible and as effectively as possible. There are many possibilities each with attendant advantages and disadvantages, for example, exhaustive methods such as depth-first search (Winston, 1975), breadth-first search (Mitchell, 1982) which have the advantage of finding the best concept description and the disadvantage of being possibly prohibitively expensive in computational terms. In contrast, heuristics may be used often with some evaluation function to direct the search as in the beam-search approach used by Michalski (1983).

Three main problem areas exist in any form of learning: namely clustering (how to subgroup), discrimination (efficient concept distinction) and generalisation. Yet there are still many outstanding problems in learning from examples due to simplifying assumptions that are often made in order to make the problem more tractable. For example, many algorithms assume no noise in the data. Simple statistically buffered algorithms as in Quinlan (1986) can be partially adapted to deal with noise. Incremental methods such as version space and the various attempts to extend the method are less adaptable (Mitchell, 1982; Kodratoff, 1988). Genetic methods learn incrementally in the presence of noise but are very expensive and inaccurate (Quinlan, 1988).

Another simplification is the assumption that the concept representational language is known in advance. If this search space is incomplete then the search will be fruitless or wrong. Another simplifying assumption is that the concept is definable rather than fuzzy; but human concepts have been shown to be fuzzy, overlapping, prototypical and usually incapable of definition in terms of a list of properties (Roth & Frisby, 1986). Many other assumptions are often made such as the concept semantics, the absence of disjuncts, etc. Further, language, we feel, seems to be at the heart of many difficulties due to the fact that

the machine does not have our appreciation of the world and hence no real understanding of whatever it happens to be dealing with.

2.1.2.6 Learning through Discovery.

Learning by example is a form of induction. The most general form of induction occurs when there is no teacher or even an explicit goal. This is known as learning by observation and discovery. The environment itself acts as the teacher. The extent of the interaction with the environment allows the classification of two major types of learning, namely, passive observation and active experimentation (Kodratoff, 1988), see Figure 2.7.

In passive observation the collection of observations are searched for regularities which can be used for classification purposes. In experimentation the system actively explores the environment inducing implications accordingly.

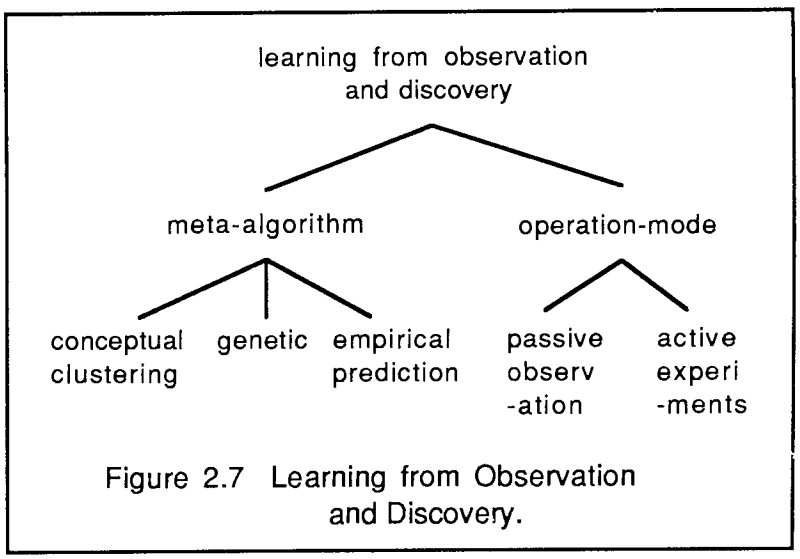


Figure 2.7 Learning from Observation and Discovery.

There are various subproblems of the field of discovery, namely, taxonomic clustering, genetic and empirical, see Figure 2.7. Clustering, called "classification" in Data Analysis) is an important problem. It involves finding a hierarchical classification that separately clusters together what is similar and what is distinct about the observations. Traditionally, statisticians and biologists have been interested in the subject and have used cluster analysis and numerical taxonomy methods.

Michalski & Stepp (1983), considered the harder problem of "conceptual clustering" with CLUSTER/2, possibly the best system of this type so far. Conceptual clustering makes explicit the descriptors and hence characterizes the clusters in order to make them easier to understand than is the case with the more traditional approaches. This is a difficult problem - remember that there is no external teacher.

The RUMMAGE conceptual clustering system uses a top down approach to create a hierarchy (Fisher & Langley, 1985). For each attribute the objects are sorted by attribute value and for each value there are frame-like descriptors which describe the corresponding objects. Attributes are selected which have the best descriptions indicating how good the concept is, where *best* is taken to mean the simplest and least similar. The method is applied recursively to create subtrees until the quality falls below a certain threshold. This algorithm is remarkably similar to Quinlan's ID3 (Quinlan, 1983).

In genetic learning the representation itself is altered possibly randomly, in a process termed "mutation". Given a boolean string the representation can be altered by, for example, "crossing over" the most and least significant bits 10010 11100 -> 11100 10010 (Caruana & Schafer, 1988). There are various such techniques analogous in prescription to biological genetics. We would suspect that there may be considerable usage of such techniques in the future since it seems to wonderfully allow the escape from deterministic representation. However, since genetic learning is a field somewhat off the mainstream of research we are unable to gauge the extent to which present-day genetic learning research grasps the essential point inherent in this suggestion. In this regard, Holland (1986) concurs with our intuition, as expressed above, when he claims that the way to escape "brittleness" is by using the more subtle effects of genetic algorithms. Holland accepts, however, that there is a trade-off between representational richness and speed. Similarly, Quinlan (1988) found that BOOLE a genetic algorithm was over 100 times slower in rule induction tests than his own C4 algorithm (an ID3 descendent). An example of integrating genetic learning and learning by example is the work of Oosthuizen (1987b).

The empirical discovery of quantitative and qualitative constant relationships between objects and variables is a branch of empirical discovery as is explanatory discovery. Such systems all really focus on generating new terms. The explanatory discovery problem (Zytkow & Simon, 1986; Langley *et al.*, 1986) of producing structural explanations of empirical laws such as kinetic energy attempts to use qualitative physics to represent processes, analogical model generation of scientific explanations and analytical learning to construct the explanations.

For example, BACON.4 is a system developed by Langley *et al.* (1983, 1986) for discovering quantitative empirical laws. Rather surprisingly it is *not* knowledge intensive, in contrast, say, to Lenat's AM system (Lenat, 1977), which is qualitative and very knowledge intensive. BACON is given the values of the respective symbolic and numerical variables concerned (such as P,V,T for the ideal gas laws) and is required to find the empirical law relating the numerical variables (e.g. PV/T = 8.3). Many nineteenth century and earlier laws in physics and chemistry have been "rediscovered" by BACON; for example the ideal gas law, Kepler's third law, Coulomb's law, Snell's laws of refraction and Black's specific heat

law. BACON defines new terms from old ones until it finds constant terms. On finding nominally independent terms it postulates intrinsic properties and has thereby postulated mass, refractive index, atomic weight, specific heat etc. The system is able, for example, to look for common divisors. This is a common technique. All empirical discovery systems focus on generating new terms. The strategy is quite simple, in contrast to Lenat's AM, with three basic heuristics:

1. Terms having near constant values are formulated into a law containing the term. This corresponds to "interestingness" in Lenat's AM.
2. Else if terms increase together then the ratio is considered, and then retry case 1.
3. Otherwise, if terms are inversely proportional then consider the product, and then retry case 1.

The system generates a depth first search tree, with periodic backing-up. It is interesting that 20th century physics seems to be beyond the scope of such systems. However the hope is that one day machines of this type may aid researchers in discovering new phenomena. Such systems may be required to provide structural explanations to account for the empirical laws, qualitative physics to represent the processes involved and analogy to generate the scientific explanations.

2.1.3 Expert Systems

Early work on expert systems, in particular the DENDRAL project by Buchanan and others as reported in Buchanan and Feigenbaum (1978), identified knowledge as the crucial component in Artificial Intelligence systems. The essential requirement was, therefore, for vast amounts of domain specific knowledge. Expert systems are designed to represent and apply the domain specific factual knowledge within some field of expertise in order to solve problems within the domain at the level of a human expert. MYCIN (Shortliffe, 1976) was the first medical expert system and so influential that it appears, at least in retrospect, to have been the first "real" expert system.

MYCIN gives advice on the diagnosis and therapy of infectious diseases. The knowledge in MYCIN is represented as approximately 400 production rules ('if condition then action' rules) relating possible conditions to associated interpretations. The rules contain "certainty factors" for a probabilistic style of reasoning. The rules are invoked by a dialogue with the user using a backward-chaining control strategy. That is, it moves from hypothesis to conclusions. MYCIN can explain the reasons, in terms of "why" and "how", for its decisions. An "inference engine" is used to effect the control strategy and this allows MYCIN to problem-solve by testing a rule's conditions against the data. The data being either initially supplied or else is requested by the system as required from the user-

physician. If a given rule condition is applicable, then the system "backtracks" to infer the truth or falsity of the rule condition for other supporting rules or data.

MYCIN was quickly followed by other medical expert systems, for example CASNET (Weiss *et al.*, 1978) and CADUCEUS (Pople, 1977). CASNET is an expert system for diagnosis and therapy of glaucoma in ophthalmology. CADUCEUS (previously known as INTERNIST) is an expert system which has considerable knowledge of internal medicine and can handle multiple disease cases.

The model of an expert system, particularly representative in MYCIN, has been successfully applied and extended to many other fields. Well known examples being PROSPECTOR for oil exploration (Duda *et al.*, 1979), and R1, which was later renamed XCON, for configuring VAX computer requests to particular user requirements (McDermott, 1981). Hayes-Roth *et al.* (1983) consider the general components of an ideal expert system. The structure of a typical modern rule-based expert system (Giarratano, 1989) includes: a knowledge base, an inference engine with possibly an agenda, a working memory containing the "facts", a knowledge acquisition facility, an explanation facility and a user interface as seen in Figure 2.8.

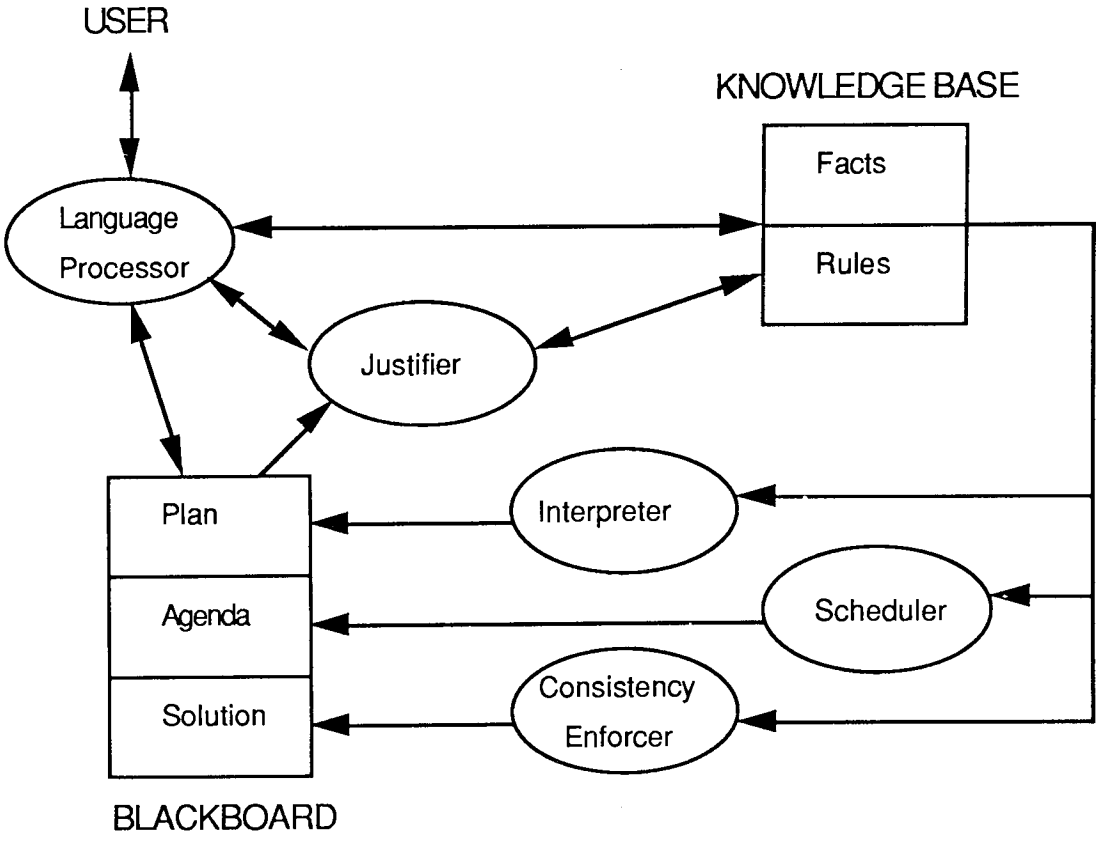


Figure 2.8 Block diagram of an ideal expert system

Today many thousands of expert systems have been built world-wide and dozens of them in a given subject area. Nevertheless, it is possible to classify expert systems as of a particular generic type. For example, the expert system may be a diagnostic system which infers malfunctions (e.g. for a car). Or it may be a planning system by designing the appropriate actions. These categories such as designing, diagnosis, planning etc. are themselves related and indicate the likely complexity of the resulting expert system. For example, a control expert system implies interpreting, predicting, repairing and monitoring system behaviour (Hayes-Roth *et al.*, 1983; Wilson, 1989). A diagram expressing these relationships between generic expert systems as a taxonomy is given in Figure 2.9.

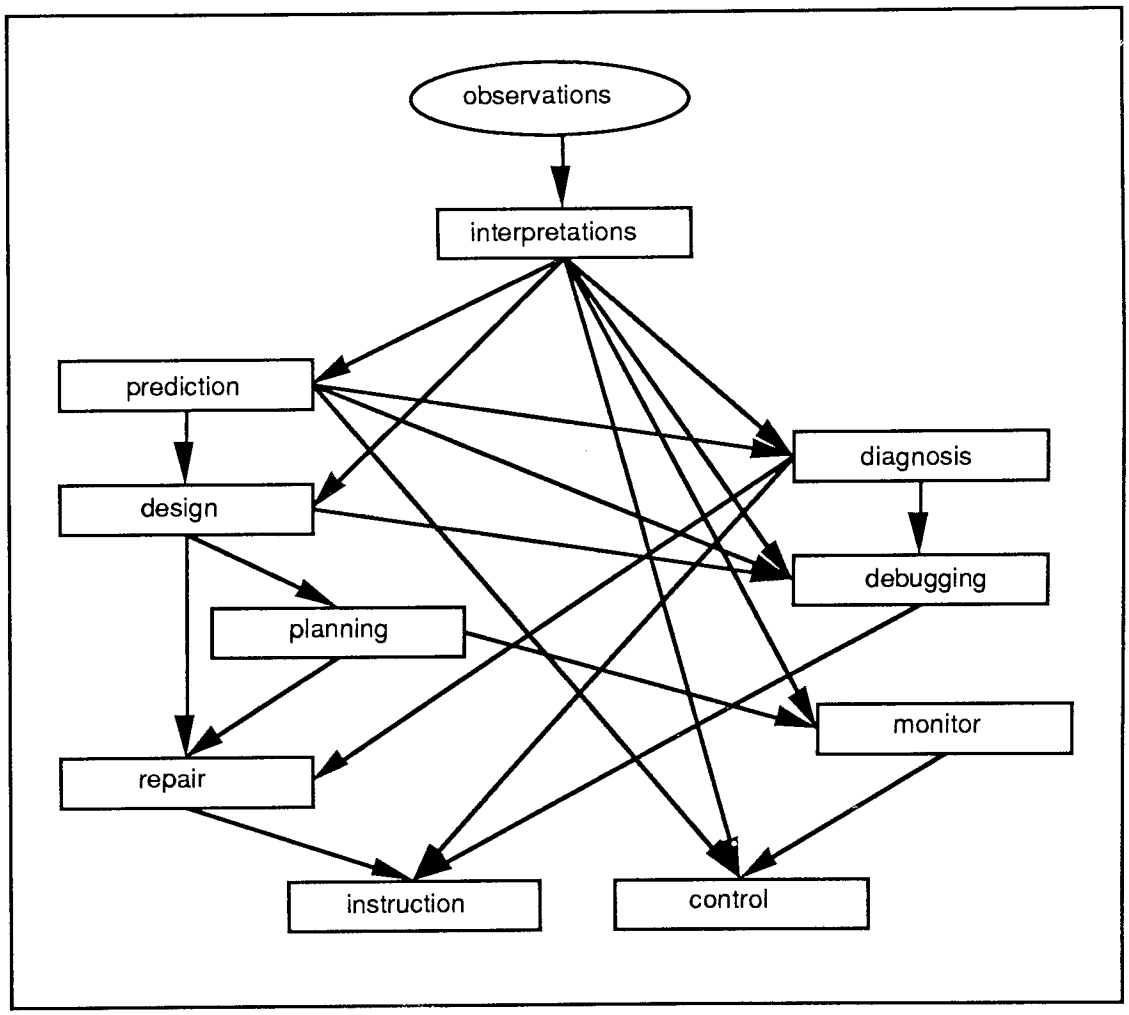


Figure 2.9 Generic Taxonomy of Expert Systems.

There are three typical present day approaches to expert systems, as applied to increasingly large domains (Turner, 1985). Firstly, there are automatic induction engines which learn by *example* from the expert. This approach is fast, typically taking 2 to 3 months, and most useful for small systems. Secondly, there are expert system *shells*, often with built-in knowledge acquisition facilities, which allow the expert himself to create a

possibly multiple-representation knowledge base. This procedure is most useful for medium scale systems and typically takes 6 months to 2 years to develop the knowledge base. The word "shell" here refers to a general purpose expert system package which has an inference engine, Knowledge acquisition facilities, user interface, etc. but is missing the knowledge base, which is built by the user.

Thirdly, for more serious and really large systems, typically taking 1 to 5 years, a knowledge engineer is required to configure the system using an *AI language* such as prolog or lisp. In larger systems a knowledge engineer, supplying the necessary state of the art science, is indispensable to act as a buffer between the expert and the expert system.

The "bottleneck" in expert systems is the knowledge acquisition phase (Michalski, 1983), since the inference engine and other parts of the system, such as interfaces to other systems, tend to be more standard (domain independent).

2.1.3.1 Knowledge Engineering.

Knowledge engineering is targeted on knowledge processing rather than information processing and involves the two subprocesses of knowledge elicitation and knowledge acquisition together with the engineering of an appropriate expert system unless a shell is available. The stages of evolution of an expert system (Hayes-Roth *et al.*, 1983) are given in Table 2.1.

Identification:	Determining problem characteristics
Conceptualization:	Finding concepts to represent knowledge
Formalization:	Designing structures to organise knowledge
Implementation:	Formulating structures to embody
knowledgeTesting:	Validating rules that embody knowledge

Table 2.1 The Stages of Knowledge Engineering.

The process is iterative and cyclic from prototype to full implementation. The first two stages comprise the knowledge elicitation phase and the latter three stages involve knowledge acquisition.

One complaint we have of this subject area is the lack of care taken with the words knowledge elicitation and knowledge acquisition. All too often they are confused. The elicitation phase involves bringing the expert's knowledge to light and comes from the Latin

elicitere, to lure forth. The acquisition phase involves the follow-on process of integrating that knowledge into the machine and acquisition comes from the Latin *acquirere* to acquire, add to or gain possession of. Thus, for example, to use the words "knowledge elicitation" to indicate the whole process is plainly careless and sloppy. Whereas it is permissible to refer to the whole process as knowledge acquisition especially in the case, say, of learning by example, where there is a minimal machine-computable elicitation phase.

Knowledge engineering is a methodology for eliciting, acquiring, representing and using computational and qualitative models of systems. These models are domain specific, exhibiting expertise in the field of the domain. The requisite knowledge is typically elicited from human experts directly and occasionally indirectly (from text). Knowledge elicitation is an investigative experimental process which may involve interviews, protocol analysis, or automatic induction from examples supplied by a human expert in order to design computational, qualitative models of expertise.

Knowledge *elicitation* is confounded by the absence of an adequate psychology (Wilson, 1989). Furthermore knowledge base representations are often finer grained than human experts are able to state (Gaines, 1988). The knowledge *acquisition* interface with experts and users is important since the encoded knowledge is of little use unless humans can understand it. Yet it is unclear how well manipulations of knowledge acquisition representations can approximate human reasoning.

The earliest attempt to build a knowledge acquisition system was by Davis (1980) in the mid-seventies and known as TEIRESIAS. Today nearly one thousand papers exist on knowledge acquisition and knowledge elicitation, which is a testimony to their importance. Furthermore, associated subjects such as machine learning and neural nets can be seen as making oblique attacks on certain aspects of the knowledge acquisition problem.

2.1.3.2 Knowledge Elicitation.

Belkin *et al.* (1988) state that the main techniques for knowledge elicitation are as follows.

1. Interviewing the expert: informally or via structured interview techniques.

2. Verbal protocol analysis: analysing recordings of experts thinking aloud as they carry out a task.

3. Observational studies: observing and recording the behaviour of experts at work as unobtrusively as possible.

Knowledge elicitation involves techniques such as card sorting, laddering, matrix generation etc. and is surveyed in Diaper (1989). Welbank, (1983) presents a sound review of the field and discusses how the different approaches each reveal different types of information. For example, one well known psychological technique is "Personal Construct theory" (Kelly, 1955), which is a top down, categorisation and questioning technique.

Welbank (1983) nevertheless cautions against the sole use of one particular technique, for example, the interview technique is commonly used, but experts have extreme difficulty in articulating their knowledge. Verbal protocol analysis is also very common (Kuipers & Kassiver, 1983), however observational techniques are the least used, since they are extremely time consuming and require in-depth analysis (Welbank, 1983). Nevertheless, observational techniques are useful for revealing what an expert actually does, information about the role of the expert, the ordering of tasks undertaken and so forth (Kidd, 1986). Cordingley (1989) reproduces Welbank's well known matrix of "Types of knowledge" by "Knowledge Acquisition methods" (she means elicitation) and discusses each technique in some depth.

LaFrance (1988) suggests a grid technique for use in discussions with experts to counteract the problem that expertise resists single category compartmentalisation and because no single question can elicit all the required information. Similarly, Minsky's "society-of-minds" theory suggests a concept of intelligence that requires the interaction of many small systems operating with an evolving overall administrative structure (Minsky, 1986). Clearly, ultimately, multiple techniques are unfortunately required (Belkin *et al.*, 1988).

Knowledge elicitation is difficult. The unreliability of experts is highlighted in a paper by Manago and Kodratoff (1987). The frailties of human nature and the inadequacies of language combine to make the extraction process highly unreliable. We tend to forget to mention negative features, for example, "... the leaves must *not* be dry ...". The *bias* of a given expert needs to be considered, but avoiding the bias by using several experts introduces the new difficulty of choosing which expert is correct when disagreements arise (Cleves, 1988). Gaines (1988) has assessed the major difficulties of elicitation. He states the following difficulties.

1. The expert may be expressing his expertise with respect to a context which he fails to define or evaluate.
2. He may be unable to express himself in language.
3. What he states may not be understandable when expressed in language.
4. What is stated may be inappropriate when expressed in language.

5 What is stated may be irrelevant or outside the terms of reference of the system under construction.

6. What is stated may be incomplete and/or incorrect.

7. What is stated may be expressed in language that is not understandable by a user or apprentice.

In short, experts find knowledge elicitation *exceedingly difficult*. However experts do find, in contrast, that it is *very easy* to give **examples** of their expertise.

Some knowledge *acquisition* systems, particularly the best of the learning by example variety, do have some reasonably solid theoretical basis. In contrast, the subject of knowledge *elicitation* is almost devoid of any theoretical grounding being pragmatic and ad hoc (Gaines, 1988; Wilson, 1989; Diaper, 1989). The result has been that there is often little difference in effectiveness of using one technique against another for most types of knowledge (Welbank, 1983; Cordingley, 1989). Furthermore, many techniques are only marginally effective in eliciting their targeted form of knowledge (Cordingley, 1989). This is more akin to stamp collecting than science.

Clearly, the subject leaves much to be desired. Only learning by example, which almost eliminates the elicitation phase, is effective. However, there are severe limitations (both practical and theoretical) to what can be achieved by learning from example with respect to the simplistic present day engines. For instance, ID3 has been found to be best suited to the early prototyping stage of building an expert system.

2.1.3.3 Knowledge Acquisition.

In general, knowledge acquisition systems can be categorised by input source, for example: an intelligent editing program (Davis, 1980), an induction engine (Quinlan, 1983), a text understanding system (Bratko *et al.*, 1985). The first takes its input from the examples directly. The second has input as data, in the form of *examples,* either from a database or directly provided by the expert. The third takes its input as typed speech from the expert in some natural language subset or from textual sources such as manuals or books (probably simplified prior to entry).

There is an important "interaction problem" concerning how the nature of the problem and the inference strategy used may strongly affect the representation. This "**bottleneck**" of knowledge acquisition, as referred to by many authors is by no means a minor inconvenience. Rather, the attempt to automate knowledge acquisition potentially requires the use of significant parts of the rest of AI. Davis (1980), Hayes-Roth *et al.* (1983) and Gaines and Boose (1988) give several examples of important problems within the field.

1. The fact that most existing systems work only with a fixed representation language developed by the designer.

2. Systems automate only parts of the implementation, testing and refinement phases of knowledge acquisition, see Table 2.1 (above). This leaves earlier phases (including the selection of the representation) to manual methods. The representation language determines the range of the describable, and hence learnable, knowledge. Therefore, since current learning systems are unable to refine their representation languages, the initial choice of representation is especially critical (Hayes-Roth *et al.*, 1983).

3. How to handle noise, that is, unexpected errors in the training and testing data, is problematical.

4 How to utilise domain specific knowledge effectively, in order to guide learning as in Explanation Based Learning.

5. How to apply old learning analogously.

6. How to learn from self-reflection, for example, self-generated practise problems.

7. Knowledge itself is not mono-typed. Each type of knowledge may then require a different type of knowledge acquisition system. This refers to the fact that the knowledge extracted by elicitation will be typed, for example, it may be factual, strategic or control knowledge, rules of thumb, declarative knowledge, procedural knowledge, etc.

8. As previously mentioned the man-machine interface forms a major aspect of any knowledge acquisition system. Such a system requires not just a graphical interface as in WIMPS, but also a sizable, robust and extensible natural language subsystem with good semantics! This request is, in itself, a very major and unsolved problem.

9. Problem difficulty discussed in Hayes-Roth *et al.* (1983), see Figure 2.10, is another dimension of major size.

10 The lack of common-sense knowledge, which is immense in size compared to the mere compilation of some tiny domain, leads to a major practical problem, namely *brittleness*. That is, unexpected system failure at the boundaries of its knowledge. This is in contrast to the "graceful degradation" of human knowledge.

11. The area of knowledge base modification, that is, maintenance, is a major, unexplored area. Expert systems, let alone knowledge acquisition systems are too new for this problem to have revealed itself yet. Nevertheless it is to be noted that, in software engineering, typically 70 to 80 per cent of the time, resources, etc. are used up in the known

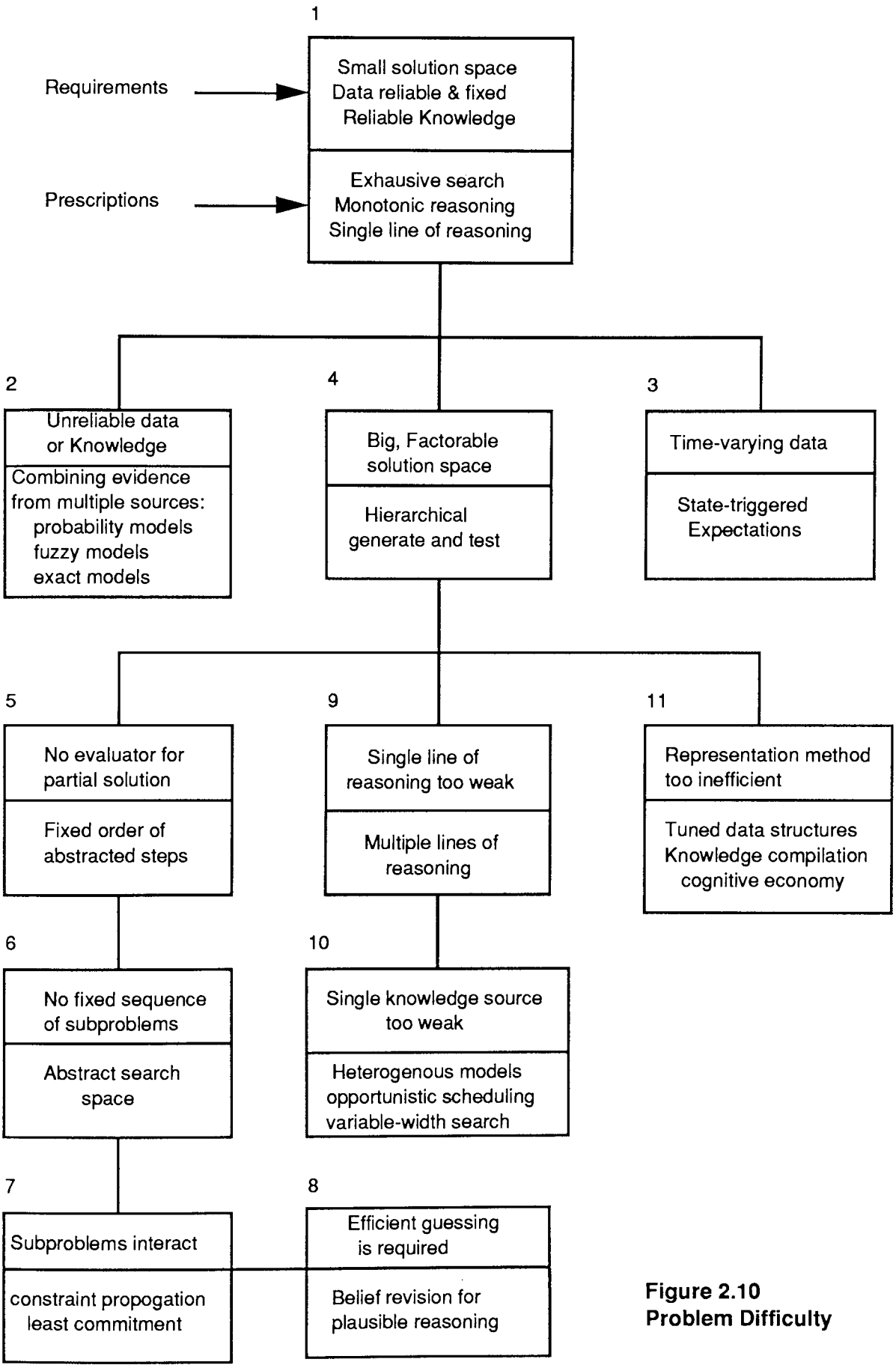


Figure 2.10
Problem Difficulty

unresolved problem of maintenance. One of the few hints of things to come in this area is the following joke, at present circulating DEC.

"One XCON system replaced 6 VAX configuration engineers, and the XCON system itself is now being replaced by 50 XCON maintenance engineers".

12. The size of the knowledge base also appears to be important. By the time the number of modules, e.g. rules, exceeds about 4000, then knowledge base additions will have become so complex that adding one further rule typically requires several days. Some reasons for this are described by Rennels and Shortliffe (1987). They cite three challenges. Firstly, modularity is difficult to obtain because new knowledge alters the old knowledge interpretation. Unfortunately, the more one accommodates this non-modular effect via links etc. the more it becomes entangled with tightly-coupled elements and thereby becomes impenetrable upon updating it. Clearly software engineering still applies to AI. We should deal in complete rule *systems* in a top down manner rather than accumulate individual rules or concepts until it all gets beyond us. Secondly, the sheer quantity of information to be handled with an exploding number of knowledge items, for example, 250,000 in the case of INTERNIST. Thirdly the lack of proper domain models. Hence automating the process would seem to be essential if we are to attempt large knowledge bases.

In a different way, Aikins (1983) has tried to fight the complexity effect by increasing the grain size in an impressive system called CENTAUR. She used frames with rules in the slots. We suspect, as does Jackson (1986), that this merely delays the onset of the problem rather than solves the problem.

13. Jackson (1986) refers to the difficulties of ensuring the production of high quality software when operating in a multiparadigm environment considering the years it took to establish the rules of software engineering. The danger being unstructured, ad hoc systems. The advantage being representational flexibility.

2.1.3.4 TEIRESIAS.

An offshoot of MYCIN is TEIRESIAS (Davis, 1980), a program that assists the expert in the construction of large knowledge bases. TEIRESIAS helps the expert to transfer his expertise to the knowledge base. TEIRESIAS is a knowledge acquisition system that acts as an intelligent buffer between the expert and the target expert system. The expert carries on a dialogue with TEIRESIAS in a subset of natural language. TEIRESIAS uses metarules (rules about the structure etc. of domain rules) to provide a knowledge acquisition tool to be used directly by the expert for enhancing MYCIN. TEIRESIAS aids MYCIN by having improved explanation capabilities, it provides a flexible knowledge acquisition tool, and has better user interface facilities than are to be found in MYCIN. TEIRESIAS facilitates the

semi-automatic acquisition of *new* knowledge for the MYCIN system. This is a shortcoming of the technique since it implies that the knowledge base must substantially exist before the technique can be applied! TEIRESIAS also uses domain independent metaknowledge and by assuming the existence of the MYCIN system is able to use rule models about rule-chaining and other system behaviour.

Statistics are used to second-guess the expert by considering variations from the norm. The process is used to detect errors in the new knowledge entered into TEIRESIAS by the expert. For example, the system uses a metarule to ensure that rules mentioning the culture site of an organism should also mention the organism's portal of entry. Some types of faulty rules are therefore detected on entry. Since the system knows about MYCIN, it can use this contextual knowledge in its dialogue with the expert. The use of metarules concerning the constituents of a rule allows TEIRESIAS to fill-in or second-guess much of a new rule for the expert. TEIRESIAS then appeals to the expert merely for verification. Meta-level knowledge also facilitates better explanations in TEIRESIAS by producing the system's "understanding" of a rule at various levels of detail, as required.

TEIRESIAS is a most impressive system. However Davis punctuates his thesis with a formidable list of problems encountered, the greatest of which by far are the language difficulties. That is, there is a *big* requirement for a reliable, advanced, natural language subsystem. This problem constantly reoccurs in some form lying, as it does, under the surface of many of the systems discussed in this chapter. The present state of natural language research can be succinctly summed up by stating that there is presently some reasonable understanding of syntactics (for the very tiny subset of any real language so far investigated) but virtually no appreciation of semantics as yet. Unfortunately, we feel that for any realistic attempt to solve these problems the vastness of the natural language task ahead *itself presupposes a reliable, advanced,* ***knowledge acquisition system!***

2.1.4 ID3

Learning by example is of particular interest, given the difficulties cited above and the advantages of learning by example (section 2.1.2.5 and section 2.1.3.2). An early and excellent system of this type in the context of developing an expert system for soybean disease diagnosis is described by Michalski & Chilausky (1980). This system achieved an impressive level of expertise by out-performing the world's expert in the subject area, even when the expert retrained on the machine's rules!

Another early example of an automatic system which learns by example is ID3 (Quinlan, 1983). ID3, however, has been the subject of a very large number of refinements (Schlimmer & Fisher, 1986; Shapiro, 1987; Utgoff, 1988; Wirth & Catlett, 1988; Cheng *et*

al., 1988; Quinlan, 1988; etc.). ID3 has also been successfully used in a number of commercial expert system tools and very many application areas. ID3 is an inductive learning engine that constructs concept classification rules in the form of decision trees from a database of examples each with an associated classification indicator (+ or -). The system is of the object-attribute-value type, where objects are described by attributes each with a number of values, as in the example below.

Attributes: height with values {tall, short}
hair with values {dark, red, blond}
eyes with values {blue, brown}

Database of examples: short, blond, blue: +
tall, blond, brown: -
tall, red, blue: +
short, dark, blue: -
...

Branches in the tree correspond to expressions of the form: **Attribute = Value**. For non-leaf nodes branches are recursively created for each attribute-value with respect to the corresponding subset of examples. The algorithm recursively builds the decision tree by choosing a good test attribute that partitions the examples into subsets, see Figure 2.11.

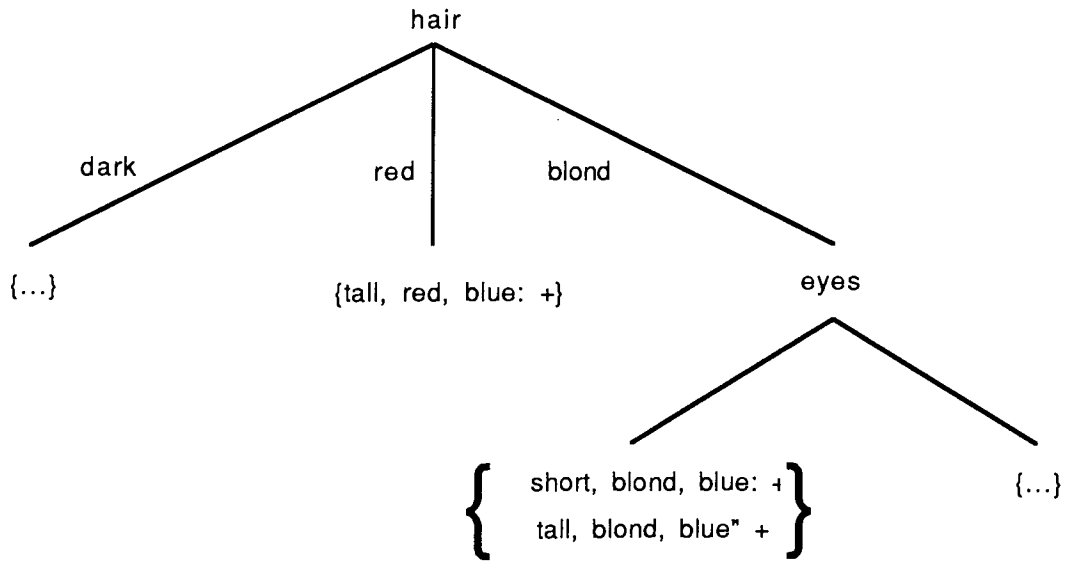


Figure 2.11 ID3 Decision Tree.

An information-theoretic measure is used to determine which attribute to use as the test attribute for a node (Quinlan, 1983). The measure relies on minimising the decision making required for classification of an instance. ID3 constructs decision trees that are relatively efficient classifiers and it generalises quite well. However ID3 has a number of

well known deficiencies: it is nonincremental, unable to handle noise, unable to cope with non-categorical data (such as height in meters), unable to deal effectively with uncertain or contradictory data and so forth. Indeed, Bramer (1987) in a *tour de force* states 64 queries and criticisms in his excellent review of ID3.

We will consider a few of the enhancements to ID3. ID3 is best suited to a static database. It's application, however, is the "real world" where noise is typical and an incremental system highly desirable. ID4 (Schlimmer & Fisher, 1986) is an incremental machine which updates the decision tree when new data becomes available. ID4 relies upon discarding the relevant subtrees upon finding that the test attribute should be replaced with a better attribute. ID5 (Utgoff, 1988) extends ID4 by reshaping the tree by pulling up the test attribute from below, which is more efficient than ID4.

ID3 uses a windowing technique to deal with large training sets. Wirth and Catlett (1988) conclude that the technique should be avoided due to it's lack of benefit in noisy domains. Various other problems with ID3 have been identified. ID3 tends to overspecialise or conversely undergeneralise due to its heuristic, hill climbing, non-backtracking search technique, it tends to produce irrelevant values for classification, it misses branches and it is biased towards attributes with a large range of values (Cheng *et al.*, 1988).

Another important practical difficulty with ID3 is that it tends to produce decision trees which are too complex to follow with ease. Shapiro (1987) discusses a nice structured inductive refinement to ID3 which is more user-friendly and which has been shown to run up to twenty times faster.

2.1.5 An Overview of the elements of Connectionism.

This area of research has many pseudonyms, for example, connectionism, PDP, ANN's, neural networks, etc. Rumelhart and McClelland (1986a) acknowledge that their "Parallel Distributed Processing" (PDP) idea that intelligence emerges from the interactions of large numbers of simple processing units has come and gone several times previously. We have already discussed two such examples in section 1.1.4. With the advent of certain new net topologies and algorithms and further a consensus of opinion that massive parallelism is a prerequisite for high performance in many AI tasks there has been a recent resurgence in connectionism.

However another reason Rumelhart and McClelland give for the present resurrection in connectionism is that symbol processing machines have failed to provide a framework for representing knowledge accessible by content (human memory is content addressable) and which can thereby be effectively combined with other knowledge. The

advantage, they suggest, being the enabling of useful automatic syntheses which would then allow intelligence to be produced.

Rumelhart and McClelland state that basic to this class of models are parallel processing, distributed representation and distributed control, which form a true "cognitive science" approach (section 2.1.1). Distributed representation implies that knowledge is not stored locally but consists of the connections among units distributed throughout the network. Hence the alternative name "Connectionism".

They further argue that software is not the whole story to the difference between AI and humans. The brain employs a more suitable *architecture* requiring the simultaneous consideration of many pieces of information as *constraints* on the processing. Each constraint may be imperfectly specified and even ambiguous yet each can play a potentially decisive role in determining the outcome of the processing.

Another basis for PDP which they suggest is that knowledge structures in AI such as rules, frames, scripts, etc. are representations which can only approximate a neural net structure. The result being that there is a big problem of *interaction* between these AI representations when attempting some generative capacity of novel situations (as we have already discussed in section 2.1.3.3). In contrast, PDP models assume information processing by the interactions of large numbers of simple processing elements called "units" or "neurons" each sending excitatory/inhibitory signals to other units. The units may stand for hypotheses, goals, actions, syntactic roles, etc. and the activations stand roughly for the strengths (weights) associated with different possible hypotheses. The interconnections between units is equivalent to the constraints between hypotheses, subgoals, etc. Hence, they feel, that there is appeal in this physiological flavour.

Rumelhart and McClelland also point out that in sequential processing the more constraints the longer the time it takes and if the constraints are fuzzy or imprecise then there is a computational explosion. Yet, they observe, *we get faster not slower* when we are able to exploit additional constraints. PDP is therefore seen in terms of the "microstructure of cognition" in contrast to the "macrostructure of thought" of serial machines.

In contrast, the intuition of great scientists is always worth noting, both John von Neumann and Alan Turing became firmly convinced of the futility of this seductive and slavish biological appeal in an earlier round of connectionism. Both men saw the way forward in terms of mathematics and abstract logic. Turing summed up his rejection of artificial neurons, simulated synaptic weights etc. in the persuasive statement "**We don't build cars with legs**". On the other hand, von Neumann's abstraction of early neural nets in terms of boolean numbers ***directly*** resulted in his first design for the hugely successful

von Neumann serial stored program machine that we have today - a little known fact (McCorduck, 1979).

The series of books Rumelhart and McClelland (1986a, 1986b), McClelland and Rumelhart (1988) form perhaps the "bible" of present day connectionism and they give many examples of the use of the sort of processing models discussed above. As an ideal example of content addressability they suggest each memory is represented by a unit with mutually excitatory interactions with units standing for each of its properties. The effect being that if a property of the memory becomes active then the memory will tend to be activated, and if the memory were to be activated then all of its contents would tend to become activated. Such a memory would not be error immune, but it would have some resilience and graceful degradation.

In non-connectionist models long term memory storage and short term memory storage are often undifferentiated or their contents may be exchanged. In PDP, the patterns, as such, are not stored, only the connection strengths between units but they are still able to recreate the patterns. This type of representation implies that the contents (knowledge) can influence the processing.

Learning also has a different perspective in connectionism. The goal of learning is no longer the formulation of an explicit representation, but rather the acquisition of those connection strengths which would allow the system to simulate the explicit representation. The connection strengths are typically adjusted *incrementally* and based on *local* information. Thus the system captures interdependences between activations. That is, it interpolates, which in turn allows "spontaneous generalisation" as opposed to the Machine Learning approach of searching for the generalisation.

The goal in connectionist systems is the acquisition of connection strengths such that the units are able to respond as though they knew the "rules." The knowledge is stored in the connection strengths of the interconnections between units. Furthermore, it is possible by increasing redundancy to provide better insulation against unit failure if pattern knowledge is not stored uniquely but rather is distributed over many connections amongst many processing units, hence producing a distributed rather than local representation. Thus the emphasis is now on pattern activation and the learning of connection strengths. Hinton and Anderson (1989) discuss various types of distributed model.

2.1.5.1 Connectionist machines.

The typical connectionist model is composed of layers of non-linear computational elements in "butterfly" form operating in parallel with the outer two layers referred to as the

input layer and output layer and the remaining layers, if any, are termed the "hidden layers", see Figure 2.12. Typically by relaxation techniques, often gradient descent, nodes produce a weighted sum of their inputs and pass the result through a nonlinearity such as a hard limiter, a threshold function or in the most general (and computationally slowest case) a sigmoid function. The diversity of these machines mitigates against a detailed review of the various algorithms - suffice to say that, at present, "back propagation" is the most popular machine.

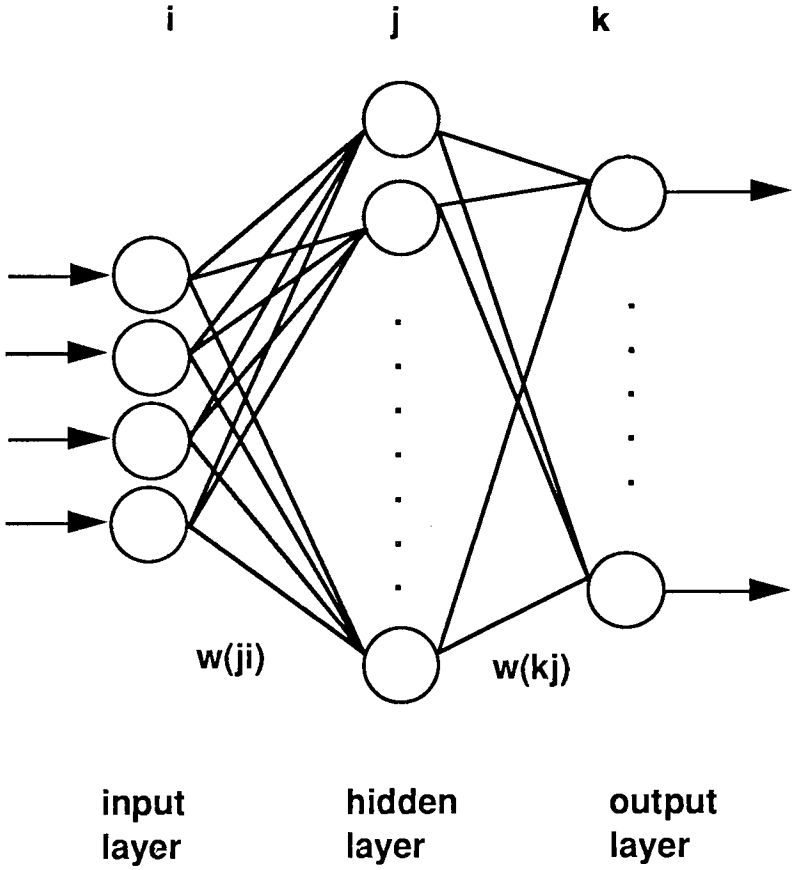


Figure 2.12 Basic supervised learning net.

2.1.6 Comparing Machines.

It is interesting to see the results of experiments comparing the performance of algorithms which go across subject boundaries. There seems to be consistent agreement between authors on such results - Machine Learning "has the edge" on connectionism. For example, Mooney *et al.* (1989) have compared the performance of the ID3 symbolic learning algorithm (Quinlan, 1986) with both the perceptron (Rosenblatt, 1962) and back-propogation (Rumelhart & McClelland, 1986a) connectionist algorithms. All three systems were tested on several large data sets from previous symbolic and connectionist experiments.

Two surprising results emerged. Firstly, that perceptron, despite its theoretical limitations performs well in practise being comparable to ID3. Secondly, back-propagation takes one to two orders of magnitude longer to train than the others. Correctness and noise tolerance of all three algorithms was very comparable, see Figure 2.13.

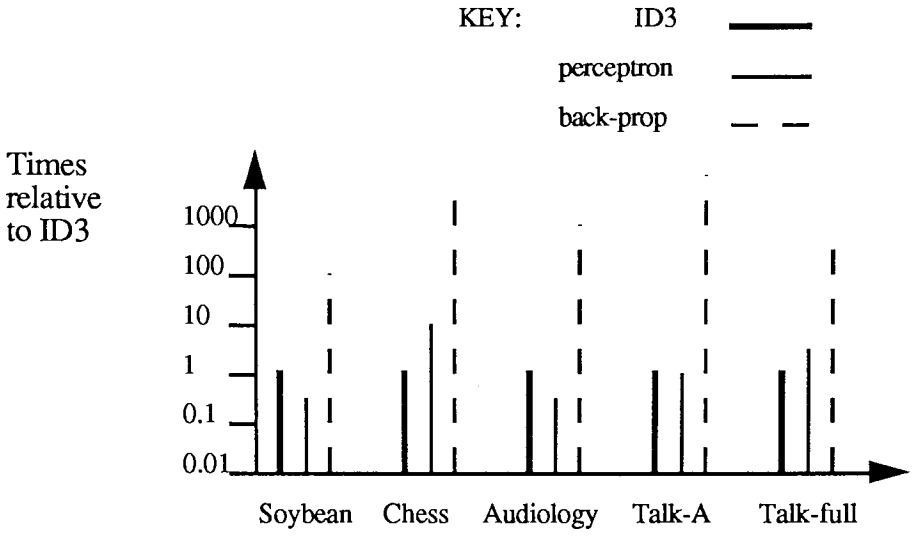


Figure 2.13 Relative training times of 3 algorithms.

2.2 INTEGRATED LEARNING SYSTEMS.

The data space of possible learning engines is obviously very large. Only a relatively few points in the totality have been identified and investigated. These isolated data points include problems such as heuristic learning, genetic learning, learning by example etc. Nevertheless it is possible to conceive of integrated learning systems which incorporate a number of such data points. Attempts to produce integrated learning systems are still rare and possibly premature. The vast majority of learning research has focussed on isolated problems such as learning by example. Examples of integrated learning systems are the series of machines based on the SOAR system by Laird *et al.* (1986), Anderson's ACT (Anderson, 1983) and explanation-based systems (Mitchell *et al.*, 1986). These machines involve some cognitive architecture which is independent of the strategy used and a set of constraints for organising performance and learning.

2.2.1 ACT.

Anderson's ACT is an attempt to model human learning performance and has been tested in several domains such as heuristic learning and grammar acquisition (Anderson, 1983). The model represents declarative knowledge in a semantic net and procedural skills

as concept heuristics in the form of production rules. There are five mechanisms for learning the rules, and the rules interact to explain the observations and improve skills:

1.	generalisation:	inductive, specific to general rules,
2.	discrimination:	inductive method for general to specific rules,
3.	composition:	produces deductive rules which fire together,
4.	proceduralisation:	produces specific versions of deductive rules and ignores that which is not required, and
5.	strengthening:	the law of practice.

2.2.2 SOAR.

SOAR is a general cognitive architecture for integrating problem-solving and learning (Laird *et al.*, 1986; Rosenbloom & Newell, 1986; Tambe & Newell, 1988; Mooney, 1988). SOAR is based on the "problem space hypothesis" (Newell, 1980). That is, all intelligent behaviour takes place in a problem space. Although we are most certainly happy with this hypothesis, it fails to mention that there are alternative perspectives.

For example, an alternative to the required problem space search is the weight space relaxation technique of connectionism. That is, we mean analogously to the sense in which algebra and geometry are alternative descriptions in mathematics. SOAR allows all decisions to be made in a single uniform way by a problem space search using "weak methods". Linking the current context to previous contexts allows the formation of a goal and subgoal hierarchy. The current context consists of four parts: a goal, a search space, a state and an operator.

SOAR assumes ***all*** behaviour can be seen as equivalent to a search through a problem space (Mitchell, 1982). Knowledge is represented as production rules for searching these problem spaces. The SOAR architecture includes only ***one*** learning mechanism which is called "chunking". The "chunk" when built becomes the production rule and thus chunking does not in itself require a search. If SOAR cannot proceed it creates subgoals which become new chunks. On finding a similar situation in future the chunk allows faster problem-solving. Chunking has been used to form macro-operators, acquire heuristics for searching and for learning from examples. Integrated learning in SOAR is therefore achieved by a method which combines learning very closely with problem-solving.

2.2.3 A Modern Classification of Learning.

The taxonomy of Machine Learning strategies is evolving with further research. Rather analogously to particle physics, it tends to expand and contract repeatedly as further

insight into the essentials of the subject is gained, as we will now see. The Machine Learning strategies defined in the texts Michalski *et al.* (1983) and Michalski *et al.* (1986) can be reduced at the topmost level to Figure 2.14. We have already considered learning by rote, analogy and instruction in sections 2.1.2.2 and ensuing sections. Learning from examples and learning from observation and discovery was discussed in section 2.1.2.5 and section 2.1.2.6 respectively and the sequel to the "from examples" and "from observation and discovery" parts of Figure 2.14 was given in Figure 2.6 and Figure 2.7 respectively.

The analytical approach of learning by deduction stands alone in Figure 2.14, which has been the perception of its lack of importance until recently. This situation is being rapidly altered, so much so, that the forthcoming Machine Learning volume 3 is expected to attempt to put learning by induction and learning by deduction on an equal footing (Kodratoff, 1988). We will now consider the reasons for this change.

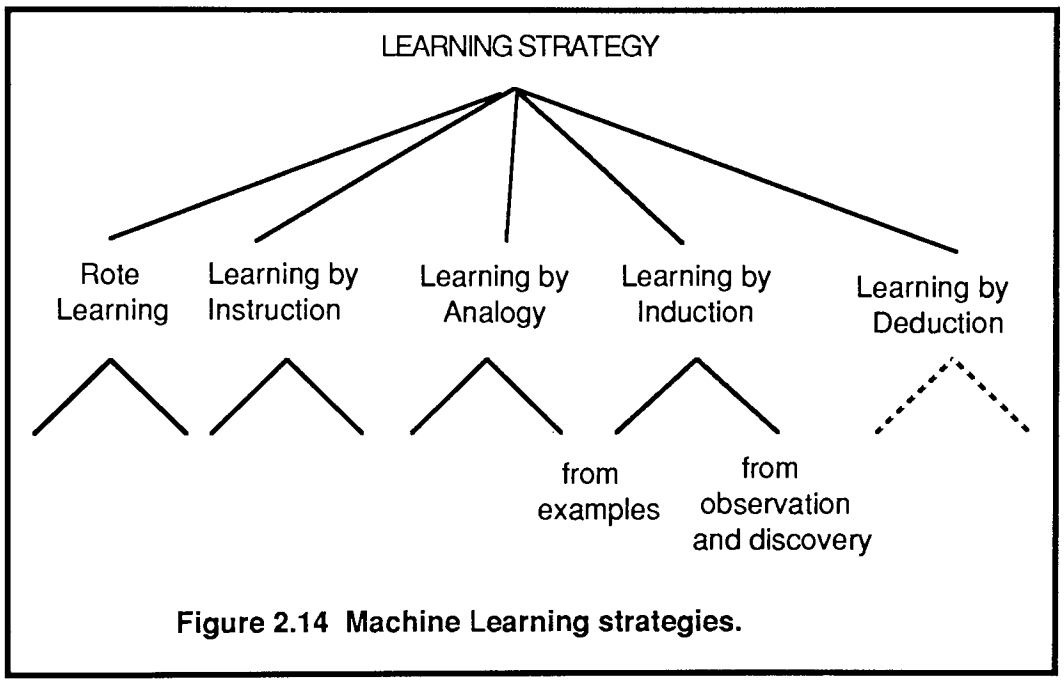


Figure 2.14 Machine Learning strategies.

Figure 2.2 plotted the extent of teacher instruction effort as against student inference effort. For a time this concept proved to be a useful theoretical perspective. Unfortunately, the "learning by X" model proved to be just too evocative. So much so, that today, there are countless examples of different types of "X" as in the template "learning by X" leading to a confusingly tangled taxonomy.

Thankfully, the emerging modern insight to the *general* problem of learning from examples has identified two contrastive approaches. This considerable simplification divides learning into one of two major strategies which attempt to approximate the human equivalents of conscious or "high level" and subconscious or "low level" learning. This dichotomy is becoming known as (the more recent) **Analytical Learning** or

Explanation Based Learning (EBL) and (the traditional method) **Empirical Learning** or **Similarity-Based Learning**, respectively. Alternatively, EBL can be seen as semantic or justifying in approach, in contrast to the syntactic or empirical approach. Thus the means of modern machine learning are:

a) deductive (analytical) techniques => truth preserving.

b) inductive (empirical) techniques => falsity preserving.

Empirical methods ***inductively*** move from *specific* data to a more *general* (e.g. structural) description. Analytical methods ***deductively*** transform *general* (e.g. functional) descriptions into another (e.g. operational) *general* description. EBL systems require considerable prior domain knowledge and attempt to generalise after observing only a *single* example and are surveyed in Ellman (1989).

Empirical learning systems (Michalski, 1980, 1983) require little domain knowledge and involve the examination of multiple examples in order to find concept features which are in common. Hence empirical learning techniques are not suitable for learning from a single example.

2.2.3.1 EBL

Two facilities which we appear to possess are the ability to learn from *single* examples (Ahn *et al.,* 1987) and our ability to accumulate and use appropriately a great deal of background, domain-specific knowledge. EBL systems (Dejong 1981; Utgoff & Mitchell, 1982) are based on the hypothesis that an intelligent system is able to learn a general concept after observing only a single example and from which it can then create "justified generalisations". In order to do this EBL systems rely on background knowledge of the domain under study. Four different tasks are usually associated with analytical systems: generalisation (Russell, 1986), chunking (Rosenbloom & Newell, 1986), operationalisation (Mostow, J., 1983) and analogy (Davis & Russell, 1987).

Analytical systems have two basic procedures (Mitchell *et al.,* 1986). Firstly, they build an explanation of the example. The explanation is constructed in the form of a proof tree, using the domain theory, that proves *why* the example is a positive instance of the goal. Secondly, they find a general principle of operation embodied in the example. That is, the general conditions under which the concept holds is determined and stated operationally (in terms of the proof tree terminal nodes). Clearly, in order to build an explanation, the system must be provided with a considerable amount of background knowledge of the domain.

The reason EBL systems are termed "analytic" is due to the fact that generalisations of the concept acquire an analysis of the example and its explanation. Ellman (1989)

suggests that the features and constraints pertaining to the example should be generalised as much as possible, as long as the explanation remains valid. Background knowledge is also required to determine which features and constraints on an example can be generalised and the generalisation will include other examples via the same explanation. Thus generalisations are said to be "justified" since they can be explained in terms of the systems background knowledge. There are a number of advantages of the analytical approach over the empirical methods (Carbonell & Langley, 1986).

1. It is not a blind leap of induction.
2. It provides a proof or "justification" for generated concept descriptions.
3. It only requires a sufficiency instance (single positive example, negative examples are not required at all).
4. It handles disjunctive concepts (as a direct result of 3).
5. It handles noisy data because the explanation process tidies up all the misclassified cases.
6. No search of a partial ordering space is required, since the explanation process provides the required description.

In contrast the analytical method has the following disadvantages compared to empirical methods.

1. It requires a great deal of domain knowledge.
2. Search is still required in the space of possible explanations.
3. Rewrite rules *really* constrain the system. The rewrite rules are functional definitions stating when the domain theory can be used.
4. Multiple explanations, if they arise, imply a search through the description space after all.
5. The explanation search space is typically very much bigger than the space of hypothesis descriptions.
6. It has less generality.
7. A theorem prover is necessary.

It seems apparent that there is a requirement for a convergence of the two methods in due time.

2.2.3.2 EBL Bias.

A crucial insight into learning is described by Mitchell (1980) in that every system which learns from examples requires some sort of bias or basis upon which to generalise and from which predictions can be made. Mitchell defines bias to be "any basis for choosing

one generalisation over another, other than strict consistency with the observed training instances" (Mitchell, 1980, p 1). That is, the search space can be constrained in many ways and these constraints are called the bias. For example, the representational bias may include only certain features of the examples or, more commonly, allow only certain forms of concept, for example, *not* disjuctions because they are troublesome! Search bias may eliminate concepts which are incomplete or inconsistent etc. or may abide by occam's razor. It is misleading to fail to make plain the bias in a learning system since it then becomes almost impossible to assess/compare/contrast it with the work of others.

Typical types of bias include using a restricted vocabulary in the generalisation language (Utgoff, 1986) and preferring maximally specific concept descriptions (Dietterich & Michalski, 1981). Bias in EBL systems may be viewed as resulting from background knowledge or a domain model. The EBL bias is towards making generalisations that can be justified declaratively by explaining them in terms of the domain model.

Several advantages result from a declarative domain bias representation according to Russell & Grosof (1987). Declarative bias is interpretable in terms of easily understood direct statements about the domain. Conversely a non-declarative bias such as a restricted language is not easily seen in terms of statements about the domain. Evaluation cannot therefore, be done purely on-sight, but only by consistently testing multiple training examples (Dietterich, 1986). Declarative bias offers the advantages of domain independence if the bias is able to be contained in a separate module, and the declarative domain model can then be an easily modifiable, uncouplable module (Dietterich & Michalski, 1981). Whereas if the bias is built into the representation and procedures of the system, as is usually the case in empirical systems, then the system is not easily modifiable and may also be domain dependent.

2.2.3.3 LEX and Version Space.

LEX 1 (Mitchell *et al.*, 1983) uses purely empirical techniques for learning concepts from multiple examples. LEX 2 (Mitchell *et al.*, 1983) is a hybrid system combining empirical LEX 1 with EBL techniques for generalising from single examples. Both LEX 1 and LEX 2 have a cycle of four main modules: a problem generator, a runtime problem solver, a critic and a learner or build-time generaliser. Problems are created by the problem generator. If preconditions are satisfied then the problem solver uses operators in a best-first search for a solution to the problem.

The system labels concepts which move the state closer to a solution as "useful". This is in contrast to its forgetting the "not useful" operators as Figure 2.15. Such restrictions reduce the number of states examined at runtime and increases the search speed.

If a solution is found, the search tree is sent to the critic. It is the critic which labels the solution search trace operators as "useful" or "not useful" according to whether the operator application leads towards or away from the solution. This classification process produces sets of positive and negative instances for each operator. The examples are then used in the generaliser to learn operator preconditions (restrictions).

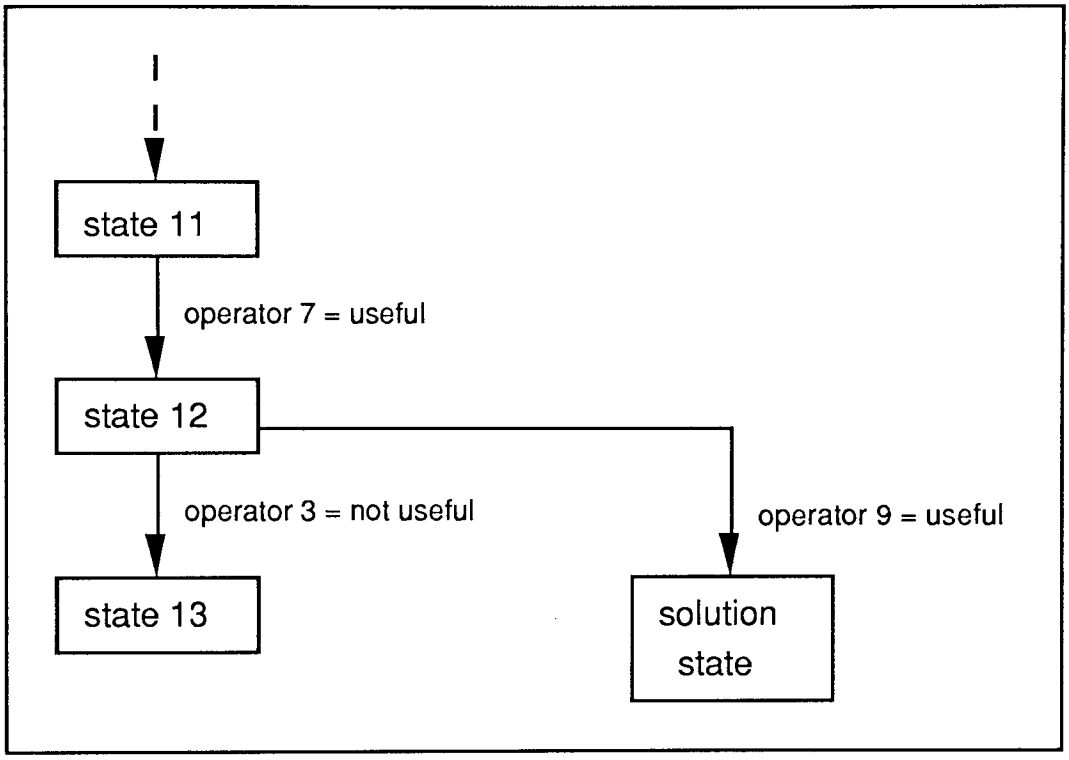


Figure 2.15 Critic labelling of the search tree.

The labelled instances are processed by the "candidate elimination algorithm" Mitchell (1978) which searches a "version space" of candidate concept descriptions interconnected by generalisation and specialisation relations defining a lattice.

On processing positive and negative examples, all candidates that are inconsistent with the critic's classification are eliminated by the candidate elimination algorithm. The algorithm maintains a maximally specific (**S**) and maximally general (**G**) set of candidate descriptions which are in-accord with the examples observed so far. This is achieved by:

1. Generalising each member of **S** just sufficiently to include a new positive example.
2. Specialising each member of **G** just sufficiently to exclude a negative example.

Since **S** and **G** move towards one another a point is reached whereupon (assuming no noise, and assuming the lattice contains the correct concept solution, and assuming there are sufficient examples available) the convergence coalesces into one sole candidate description for the concept being sought. Hence, version space combines the specific to

general and the general to specific approaches to learning from examples (section 2.1.2.5). The S and G descriptions are not hypotheses (as is usually the case) but rather are seen as constraints or boundaries on the current set of hypotheses.

The advantages of version space are: that it converges quickly, can incorporate new instances easily, the sets S and G summarise the information thus far (hence there is no requirement to keep previous instances) and it knows when it has learnt the concept since S and G then merge. The disadvantages are the very strong bias: it assumes conjunctive descriptions only and an absence of noise. This machine is beautifully pure in concept, but unworldly! For example, if there is any noise **S** and **G** may just simply pass one another and keep going! Subsequent attempts by Mitchell and others to modify the algorithm have failed to avoid this problem (Kodratoff, 1988).

LEX 2 provides the learning modules with further knowledge (for example the "goal" of the learning process) in order to *enhance the effectiveness of the generalisation.* This, together with single example learning is the main advantage of EBL techniques. That is, it generalises the operators to a larger class of states, typically by replacing constants by variables - a standard technique (Michalski, 1983) as discussed in section 2.1.2.5. Hence in theory, at least, LEX 2 learning should converge faster than LEX 1. If so, this would indicate a stronger "bias" for inductive learning. However the effect only works in the initial stages of learning and newly acquired heuristics soon lose their effect in contributing towards any enhanced learning ability (Mitchell, 1983).

The difficulty is due to the fact that the learnt heuristics can only improve some aspects of the system's performance. The heuristics (or preconditions) aid in deciding operator applicability for some given state but not in deciding which state to chose for expansion. The limitation of being unable to decide on which state to expand leads to what Mitchell calls the "wandering bottleneck" problem (Mitchell, 1983). Avoidance of wandering bottlenecks implies the system must have the *ability to automatically formulate its own learning tasks* (Keller, 1987). So in order to affect performance further, more domain knowledge is required, but this is a recursive request! One might suspect improvements may be proportional to the extra knowledge programmed in - although such a measurement would be difficult!

EBL generalisations can be combined with empirical methods. For example subconcepts found by LEX 2 are sufficient but not necessary conditions for concept membership. Therefore subconcepts, as generalised positive instances, exist in the lattice and can be processed by the candidate elimination algorithm as if they were actual positive instances.

We will not discuss LEX 2 in detail, however one aspect is of considerable interest. LEX 2 removes references to operator applications in order to obtain candidates expressed in terms of the example state by a constraint back-propagation procedure (CBP) (Utgoff, 1986). This is equivalent to calculating the "weakest precondition" (Dijkstra, 1976) and performing "goal regression" (Nilsson, 1980), and is analogous, at least in operation to error back-propagation (EBP) or, more commonly, back-propagation (BP) as the standard relaxation technique in connectionism (Wasserman, 1989; Rumelhart & McClelland, 1986a, 1986b; Anderson & Rosenfeld, 1989; McClelland & Rumelhart, 1988).

Back-propagation could provide common ground for combining the strengths of Machine Learning EBL and Connectionist EBP (Gangly, 1987). A "coalesced machine" of possibly this form was stated in the introduction to be the aim of our project. Hence we have one possible approach for this present work. However, the difficulties of EBL which we will now highlight mitigated against this idea.

2.2.3.4 EBG.

An "**Explanation-Based Generalisation**" or **EBG** formalism has been proposed (Mitchell *et al.*, 1986) in order to capture the essentials of the emerging EBL systems. EBG is partly domain independent. EBG has two parts:

(A) the EBG problem:

Given:

1) a goal concept,
2) training examples,
3) a domain theory,
4) an operationality criterion (which specifies the types of concept that are operational with observable or evaluable predicates)

then find concepts that are:

1) generalisations of 2) above,
2) satisfy the sufficiency condition for the goal,
3) satisfies the operationality criterion.

(B) the four interpretations:

1) generalisation: training examples -> operational concept descriptions
2) chunking: domain theory -> concept membership test rules
3) operationalisation: goal concept -> operational concept descriptions
4) analogy: training and test examples -> test example classifications.

2.2.3.5 Evaluation of EBG/EBL.

Dejong and Mooney (1986) criticise EBG as deficient in several areas and state that EBG:

1) undergeneralises,
2) cannot generalise the domain theory,
3) cannot generalise the structure of the explanation,
4) has deficiencies in the operational criterion,
5) fails to note whether the information source is human or system built.

EBG requires a priori a domain theory and a goal concept. Hence (Ellman, 1989) probingly queries:

1) *are training examples necessary for EBG systems?*
2) *do EBG systems only learn things already contained in the domain theory?*
3) *in what sense can EBG be said to improve an intelligent system?*

The first question results from observing that, if the domain theory is capable of explaining the example, then the same theory might be sufficient for generating the example in the first place! Hence the training example appears redundant! The second question results from considering that the system creates the concept recognition rules by deduction from the domain theory, hence what new thing is learnt?

Dietterich suggests that a system can perform "knowledge level learning" only when there is a change in the "deductive closure" of its domain theory (Dietterich, 1986). The deductive closure of a set of axioms being defined as the axioms themselves, plus all facts derivable from the axioms. But the concept membership test rules created by EBG are all contained in the deductive closure of the initial domain theory. Hence, Dejong deduces that EBG does not change the deductive closure of the knowledge base and so does not perform knowledge level learning!

We wonder how things might be otherwise? Possibly genetic learning or random mutation may escape the dilemma by arbitrarily altering the representation (for example the work of Oosthuizen, 1987b), but the price may be high - most systems altered in this way would no doubt be useless. The possible effect can be seen in another EBL difficulty.

Further problems with EBL systems are the creation of rules that are rarely useful (Minton, 1985; Fikes *et al.*, 1972). The useless rules consume storage space and degrade efficiency by attempting to apply useless rules. Although EBL can improve performance over empirical learning Minton has shown that it can also *degrade* performance by

uncontrolled chunking (Minton 1985). Minton also showed that performance can only improve when there are heuristics to decide when to create and retain chunks (Minton 1988). This, however, implies that common sense contextual knowledge is also required! Similar results are obtained by others (Tambe & Newell, 1988; Markovitch & Scott, 1988). The latter study showed improvements result from random or selective deleting of excess macros somewhat in-accord with the genetic or random-mutation learning observation above.

As Ellman (1989) states EBG raises questions about the *value* of EBG which may imply premature formalisation. The results of EBL depend critically on the representation of the domain theories and explanations. For example, (Gupta, 1988) has identified cases in which the generality of learnt rules depend upon the details of the domain theory representations, such as the grain size (Braverman & Russell, 1988). There is widespread agreement that representation is critical, clearly identified by Minsky in McCorduck (1979), but the elements of a good representation in EBL are unknown (Ellman, 1989).

EBL systems such as LEAP (Mitchell, 1985) use EBG methods to learn by watching a human expert problem-solving. This knowledge acquisition must not make inordinate demands on the expert's time. For example, if building the initial domain theory is comparable in size to building the finished expert system, then the advantage of using the knowledge acquisition tool shrinks recursively since the initial domain theory can be seen as expert knowledge about the domain, itself requiring a knowledge acquisition tool ...!

Interfacing with an expert and its inherent natural language challenge, raises big problems with regard to learning engines in general. In conventional engineering it is sufficient to find the crucial elements of the effect required. For example, the usefulness of the aerofoil shape for flight. In contrast, a learning engine must not *only* employ this step but also it has to *effectively* interface with humans in order to communicate its results. And this we know is very non-trivial.

EBL systems assume completeness of the initial domain theory in order to improve the domain theory. Ellman (1989) gives four dimensions of initial theory defects, implying the possibility of incomplete domain theories, incorrect domain theories, inconsistent domain theories and intractable domain theories.

1. Completeness: Does the theory entail at least one positive or negative classification for each example in the domain?

2. Consistency: Does the theory entail at most one positive or negative classification for each example?

3. Correctness: Are all the predictions entailed by the theory correct?

4. Tractability: Can explanations of all examples be constructed without exhausting specified time and space resources?

All these difficulties and deficiencies outlined above, particularly the natural language semantic problems underlying many of these problems mitigated against a high level "coalesced machine". Hence we now turn to consideration of extant low level, empirical engines with sufficient commonality to form the basis of a coalesced machine.

2.3 HYPERCUBE MACHINES.

We have considered the elements of the three fields of interest, namely, Machine Learning, Knowledge Acquisition and Connectionism including several particularly interesting machines in section 2.1. In section 2.2 we highlighted some of the difficulties inherent in the proposed integrated approach together with some of the deeper aspects of learning such as bias. Again several machines were of interest particularly the lattice representation aspects of the version space approach. Finally, in this last section we discuss machines from the three areas of Machine Learning, Knowledge Acquisition and Connectionism which exhibit lattice-based commonality. In these cases the lattice forms a particular structure in multidimensional space known as a hypercube.

2.3.1 Graph Induction.

Oosthuizen (1986) describes an associative set model based on the generalisation hierarchy ideas of Lebowitz's UNIMEM system which we will briefly describe first. Lebowitz (1986) automatically creates concept hierarchies in order to construct a knowledge base. Examples represented as sets of features are automatically built up into a generalisation hierarchy using similarity-based methods. UNIMEM centres around the principle of Generalisation Based Memory (GBM). Examples of concepts or situations are compared with existing generalisations in the hierarchy, starting with the most general ones and proceeding towards the most specific until the most specific generalisation is found that best describes the new instance. Before inserting it in the hierarchy, a check is made for stored generalisations which have features in common with the new instance, in which case, the instance is generalised sufficiently. UMIMEM, therefore, allows the characterization of new concepts to be formed on *regularities* discovered within the example descriptions. UNIMEM has the following features:

1. Learns by observation, not guided by which concepts to define/compare.
2. Incremental learning, one example at a time, produces the best generalisation so far.
3. Example features have the format of property/value pairs.

4. Pragmatic:
   - a) generalisations do not necessarily cover all examples!
   - b) generalisations may overlap!
   - c) generalisations are modified/eliminated if:
     - 1) contradicted,
     - 2) insufficiently confirmed.

In his paper "A Paradigm for Automatic Learning" Oosthuizen describes a method which extends UNIMEM and is based on a *set theoretical approach* to knowledge representation (Oosthuizen, 1986). Oosthuizen argues the case for his approach as follows. Firstly, he states that machine learning is still a relatively immature technology, and that relatively few algorithms have been implemented as commercial products. Oosthuizen explains the reasons for this state of affairs as:

1. methods are not robust.

2. methods assume either:
   - (a) domain samples with a very low noise level, or
   - (b) a well developed domain theory
     (which may minimise the need to learn)

3. methods often involve *complicated* algorithms. This implies:
   - (a) requires user guidance and iteration (so not autonomous),
   - (b) increasing algorithmic sophistication often increases domain dependence.

In contrast, Oosthuizen states that his contribution "... is in the area of representation, facilitating an automatic learning method that is remarkable for its simplicity and implementable on highly parallel hardware," (his underlining). We most definitely approve of Oosthuizen's intuition, clearly expressed in the preceding sentence. Oosthuizen (1986) describes his 'associated set knowledge representation model' (ASM) and its inherent transformations as a "semantic" network model. That is, ASM is a directed graph consisting of nodes connected by directed arcs. Each node P is a set, as defined in the set theory:
$\{ x / P(x) \}$ where P is a predicate over x.

Nodes are interpreted as a set with a discrete number of members with an associated common property. Directed arcs are the only type of network link that is allowed and represents set subsumption. Oosthuizen states that this corresponds to the well known inheritance hierarchy incorporating classes and subclasses (Tourzetsky, 1986) in which terminal nodes are one or more individual entities or set members.

Arrows in the graph point upward from subclass to class. Thus the set represented by the lower node is contained in the set represented by the upper node. Oosthuizen states that by considering individual members as (terminal node) sets containing that particular member only, the network then reduces to a uniform system of nodes representing sets only and a uniform set of links representing subsumption only. The set intersection A of sets B to E is represented by arrows as Figure 2.16.

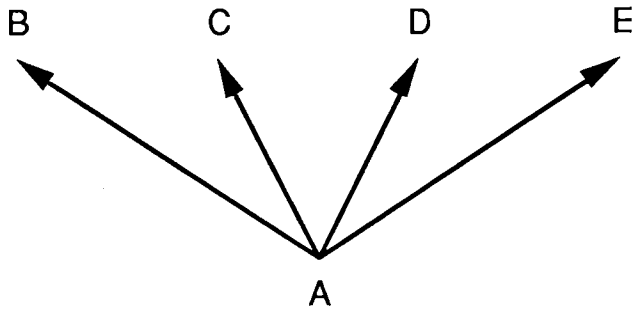


Figure 2.16 Set Intersection

The union of sets B to E in set A is represented by arrows as shown in Figure 2.17.

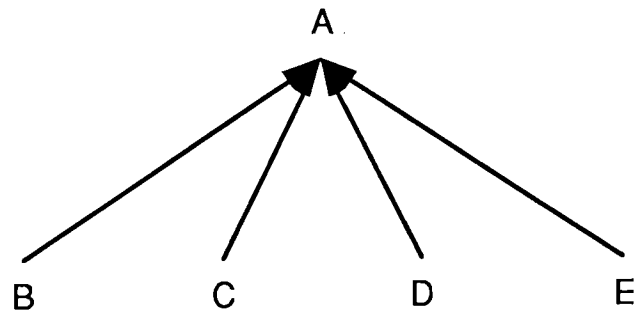


Figure 2.17 Set Union.

Concepts, described by the conjunction of attributes, are represented by the intersection of sets being contained in another set, as shown in Figure 2.18.

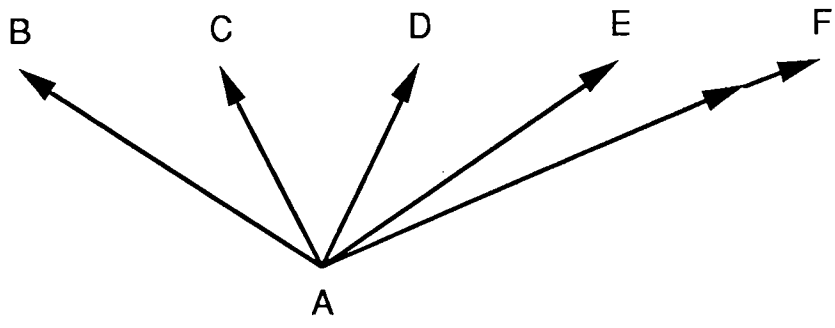


Figure 2.18 Containment

The double headed arrow indicates the intersection of B to E (i.e. A) is contained in the set F. (i.e. F implies a *concept* as opposed to just any intersection of sets). This we feel is ad hoc since it breaks with an otherwise near orthogonal structure.

Oosthuizen states that ASM relations are all transitive. This has no effect on the expressive power of the network because the definition of a set (as above) implies that the relationships between sets merely state whether certain objects present in a particular set are present in another set or not. In ASM the set relationships (subsumption, intersection and union) reduce to a one by one count and comparison of members present in each of the sets involved. Non-transitive relations (e.g. 2 place predicates) are also expressible in the network in a notation closely resembling Tourzetsky's excellent work on inheritance (Tourzetsky, 1986). For example, John loves Mary is represented by a set of "things that love Mary" containing "John". This set is formally: love(MARY) where MARY is the label of another node in the network, itself a member of the set: loved-by(JOHN).

The expression before the parenthesis is to be regarded as a function, having a unary predicate node in its domain and each generating a possible corresponding node. Each function generates such a network forming a generalisation hierarchy and explosion of nodes. Oosthuizen states that the basic framework of ASM has the important characteristics:

1. very simple formalism,
2. all information is stored as part of a transitive ("semantic") structure,
3. ASM uses only the three (above) fundamental concepts of set theory.

Additional restrictions are placed on the network structure in order to:

a) enrich its structure
b) enhance the potential for reflecting semantic nuances.

In ASM, any member of sets may have only one node representing their intersection. Hence the "distributed intersection" (DI) as Figure 2.19 is disallowed:

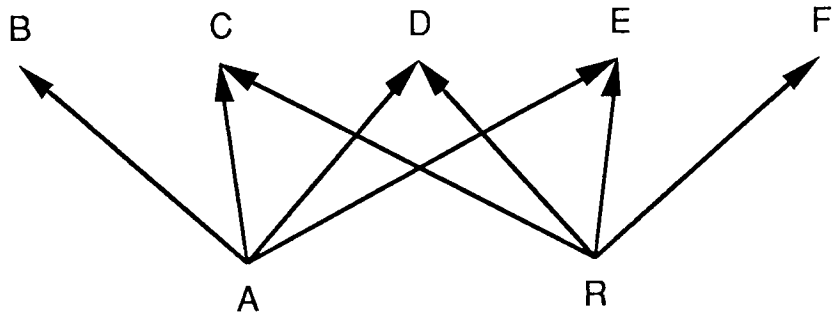


Figure 2.19 Distributed Intersection.

This restriction entails some network transformations. For example, nets such as Figure 2.20 (a) are changed to type Figure 2.20 (b) in order to remove the DI.

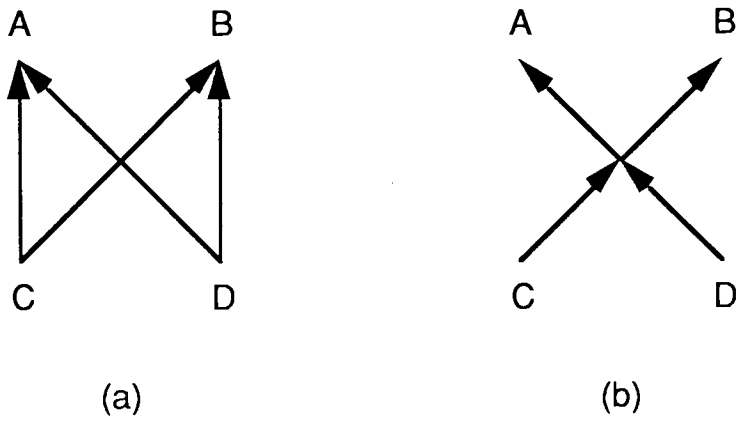


Figure 2.20 Removing the DI.

The first transformation corresponds to observation based characterization, based on regularity recognition or generalisation in Lebowitz (1986), or forming minimally-redundant data structures in Elgalal (1985). Two further transformations correspond to some of the generalisation rules of Michalski (1983) yet no new theory of knowledge acquisition is required. Thus the application of the knowledge base housekeeping procedures as transformations imply learning. Learning activities become integrated with the updating procedures. However for the worked examples he gives the method appears to be increasingly opaque for larger problems with exceptionally tangled network hierarchies.

In Oosthuizen (1987a, 1987b) he changes the algorithm name from ASM to GRAND (Oosthuizen, 1987a) standing for graph induction, and an enhanced version SUPERGRAN (Oosthuizen, 1987b) allowing genetic reformulation of the data in parallel with the Graph Induction; and he simplifies his algebra to a boolean schema notation accessible to the genetic machine. However the main changes are the introduction of a genetic algorithm to work on the schema bit string to induce novel concepts (this appears to be a strong technique), and the investigation of the use of a parallel set closure machine developed within the department for fast induction. Oosthuizen states that apart from the absence of the null vertex his machine is structurally and semantically equivalent to Ganascia's Hilbert hypercube machine, CHARADE (Ganascia, 1987a). The advantages of this representation are:

1. it integrates empirical and explanation based learning,
2. hence handles both small and large numbers of examples,
3. learning is a *side-effect* of updating procedures, cf. algorithmic,

4. the model can be implemented on specialised parallel hardware,
5. hence search problems are highly reduced.

With regard to integration, Oosthuizen states that existing methodologies may be implemented on, and integrated with this work resulting in a surprisingly powerful new method of learning. Tests show that it out-performs ID3 (Quinlan, 1983) on ID3's published examples in the case when the number of examples is small. This is not too surprising since ID3 does not optimise well.

In a further paper Oosthuizen and McGregor (1988) discusses his graph induction engine in terms of knowledge base normalisation (akin to database normalisation) for optimally integrated rule base construction; that is, where integration is optimal and terms appear once only. Inductive learning is discussed in terms of the lattice operations 'meet' and 'join' and 'greatest lower bound' and 'least upper bound' on the normalised knowledge base. This paper gives a connectionist perspective which we summarise as follows. The graph produced is a sublattice from an examples layer to an attribute-values layer with the evolving layers in-between forming concept hierarchies. The semantics of the network are such that the nodes are regarded as sets "covering" the objects below them and the arcs are regarded as subsets.

For implementation of a fast machine in hardware he cites the possibility that his set machine will run on special purpose hardware, the Generic Associate Array Processor, GAAP, (under construction) which will dynamically construct networks to process set data via set operations in parallel. With regard to his present environment, set closure on a SUN 3 is stated to be heavily problem dependant (hardly a surprise) and from one millisecond to half a second for *one* closure on problems tried so far. Storing relationships explicitly is stated to incur higher computational costs and higher storage requirements than Ganascia's CHARADE system (Ganascia, 1987a).

2.3.2 The Hilbert Cube.

Ganascia in a series of papers (Ganascia, 1987a; 1987b; 1988) is mainly concerned with the problem of rule *systems* - as was Davis (1980). Davis tried to use a substitute for the complete natural language interface with the expert that he really required, in order to incrementally acquire a consistent rule system. However, the traces from Davis's machine at runtime amply show the difficulties, which he fully acknowledged, resulting from his brave attempt to use such a paucity of linguistic structure.

Ganascia, on the other hand, attempts to capture a rule *system* satisfying operational criteria via a set of mathematical axioms, prior to the use of his Hilbert

hypercube representation. But again, the paucity of a few axioms of mathematics to capture the diverse structure of language seems evident. The intent behind Ganascia's system known as CHARADE is the development of an automatic learning system which can learn a *consistent* rule *system* from:

a) A descriptive language.
b) A set of axioms about the language semantics.
c) A set of examples expressed in that language.

The most interesting part of the system is the data structure used namely a Hilbert hypercube which is an orthogonal boolean lattice. A Hilbert hypercube is an n-dimensional cube or hypercube where n refers to the size of the learning set. Each axis of the cube is associated with an object. Each vertex then corresponds to a set of objects and set membership can be determined by projecting the vertex onto the corresponding axis.

The organisation of the rule base is achieved by arranging all terms into this hypercube or lattice structure. The lattice structure is preserved implicitly by propagating relationships between terms and only storing the actual rules obtained. This contrasts with the above set closure system Oosthuizen used where all relationships are stored explicitly thereby gaining a complete inventory of all dependencies between terms or groups of terms - but at a higher computational and storage cost. A second cube is created for the set of features or descriptors as Figure 2.21.

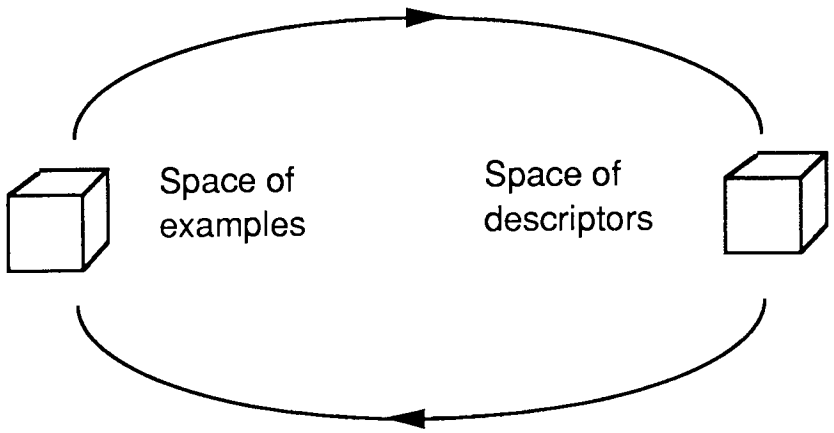


Figure 2.21 The example descriptor dual hypercube system.

Functions are defined which map these two hypercubes onto each other and remove redundances. Objects and features are clustered into sets which are related by inheritance relationships in the structures that form the lattice. Ganascia generalises atomic descriptors to intervals of values and he uses statistical rules to derive approximate probabilistic rules.

This is a noise containment approach in that probabilistic rules are used to identify outliers and alter the graph accordingly.

As Ganascia states, a big problem exists between mere rule juxtaposition and a complete, efficient rule *system*. This gap CHARADE tries to bridge by detecting logical or statistical *regularities* in a set of examples. The work is a strong contribution to the field in this area - in effect, it amounts to a rule base *maintenance* approach from the outset, something a software engineer would appreciate. For example, he states that modularity is insufficient, operational criteria such as lack of redundancy, lack of cycles, consistency, completeness, etc. are necessary (which is all "good stuff").

Where we feel that the work begins to come apart is that point where, when faced with severe difficulties, ad hoc solutions are sought. In this respect Ganascia seeks heuristic solutions at times. The system then compromises on the purity and strength of the representation, degrading its ready mathematical formalism. The resulting *"triggerable learning parameters"* then serve to highlight this weakness. Ganascia bravely tried to cover too much ground. Nevertheless, the system is as a whole most impressive.

In a further paper Ganascia (1988) outlines the learning *bias* of the CHARADE system in terms of the semantics of the representational formalism, the description language and the learning assumptions. On representational formalism the machine is stated to work in propositional logic thereby precluding constant variables in the generalisation process. The generalisation is limited to the intersection of descriptors as triplets of the form:

< Attribute > < Selector > < Value >

for comprising formulae. The two description languages are the source language (for examples) and the target language (for generalisations). With regard to learning assumptions he states that the completion of the example descriptions can be used to add new descriptors checked by the expert.

This paper although mainly repeating the work discussed in earlier papers offers further details and new angles on the system. This is particularly useful since Ganascia's English is not strong. We do, however, feel that there is a difference between the important task of highlighting the bias of a system and merely relating the complete system; whereupon the bias may "get lost" in the detail. In summary, CHARADE is a very novel system identifying a thoughtful creator, possibly suffering a little from ad hoc additions for completeness. Its thrust is less connectionist than Oosthuizen's Graph induction and more an analytical, machine learning approach.

One solution may be not to "try to run before we first learn to walk". The simpler, preceding problem of learning to acquire concept descriptions from a set of examples and

counter examples should, we feel, be first considered. Such a low level machine, if adequately researched and very widely applied, might then limit the space for renewed attempts at the much more difficult problems attempted by Davis and Ganascia, but even that, we feel, is still we feel a very long term aim. Higher level learning is seductive. Since we quite naturally think and reason about things, progress would appear to be fastest when we concentrate our efforts on machines which similarly learn at the higher levels. Examples of such machines abound throughout AI generally and, in particular, in learning.

Such systems are *uniquely* characterised by an **analytical** rather than an **empirical** approach. But the essence of the analytical approach to learning is that it necessarily implies a prior analysis of the problem. Yet the inescapable result of analysing everything that the machine needs to know in order to cope with non-trivial, real-world problems, necessarily tends to entail a tendency towards an infinity of domain specific programming, such as knowledge acquisition; but *learning* is just what was originally postulated to overcome just this very problem. Often the solution suggested for solving the domain specific deficiencies is yet another round of analysis!

Thus, learning is required to overcome the immense difficulties of the analytical approach. The seductive solution being analytical learning. This analytical learning then implies analysis, which can only be solved, due to the enormity of the task by learning, ...! With analytical learning as our only paradigm we necessarily have a tautology. The only escape seems to be some measure of that much less interesting tortoise, empirical learning. In effect this makes explicit some of the impetus behind the modern connectionist approach. We would expect it to be a very long road from the lowest non-symbolic, empirical levels to even the lower reaches of the higher symbolic levels. We consider this point in more analytical detail in the next chapter.

2.3.3 Brain State in a Box.

"Brain State in a Box" or "BSB" is one of the half dozen or so best known basic connectionist architectures. The BSB model (Anderson, 1977; Anderson et al., 1989; Anderson & Mozer, 1989) is similar to the linear associator. BSB has a maximum and minimum activation value associated with each unit. Activation values are in the interval [-1, +1]. The representation is a hypercube with the centre of activation at the centre of the hypercube. Activation can recycle by positive feedback. Since positive feedback is inherently unstable it is limited. For example, Anderson & Mozer (1989) have an activation rule given by:

$$x_j(t+1) = x_j(t) + \Sigma\, w_{ij} x_i(t)$$

where: $-1 >= x_j >= +1$; $x_j = -1$ if $x_j < -1$; $x_j = +1$ if $x_j > +1$.

Activation at time t + 1 is the sum of the state at time t and the activation propagated through the matrix within the interval [-1, +1]. The result is that the system converges to a state in which all the units are of maximum or minimum value. Therefore starting from any given point in the hypercube the function ends up because of the limited positive feedback saturation effect in a "corner" or vertex of the hypercube as Figure 2.22. Analogously to Ganascia's CHARADE system the axes correspond to the activation of the first, second, etc. units. In contrast to CHARADE where only the vertices are involved in the computation, each point in the space in BSB corresponds to a possible state of the system and the hyperfaces and vertices delimit the space of computation.

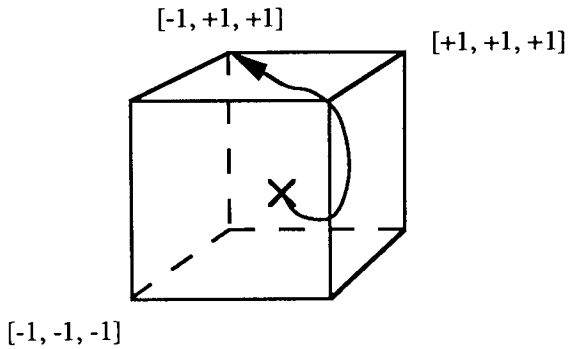


Figure 2.22 Positive feedback in BSB.

Anderson & Mozer (1989) employed a simple Hebbian learning given by:

$$\delta w_{ij} = \alpha\, x_i x_j$$

Error correction has also been applied to BSB whereby the input is used for teaching as well as the source of activation according to the rule:

$$\delta w_{ij} = \alpha\,(t_i - x_i)\, x_j$$

such that t_i is the input to unit i and x_i, x_j are the activation values of the system after convergence onto one of the hypercube vertices. Anderson says that because all points within the space are "interpreted" identically by the system any initial state occupying a given region of the hypercube will end up in the same corner which thus puts "... noisy inputs into saturated form ..." In contrast, Anderson *et al.* (1988) by:

> "...using Hebbian "antilearning" (outer product learning with the opposite sign), it is possible to destabilise a stable state and make the system move to a new state."

They further state that BSB has also been used for probability learning where it does a "reasonable job" of accounting for various observed experimental effects in both man and animal. Recent work on BSB has focussed on concept formation abilities (Anderson,

1986) and hopfield-like energy minimisation (Golden, 1986). Golden produced formal mathematical proofs showing, for example, that it is a gradient descent algorithm which minimises a quadratic energy function. The psychological plausibility of the model is considered in Wisniewski and Anderson (1988) showing, for example, the acquisition of multiple classes of objects and categorisation of noisy, incomplete and degraded stimuli. The system demonstrates a measure of categorisation certainty as the number of iterations to saturation and can easily detect deviations from its expectation. Clearly BSB is powerful and interesting and is an on-going research topic. The primary application of BSB is the extraction of knowledge from data bases. A limitation of BSB is its one-shot decision making - it has no iterative reasoning. BSB is similar to the "bidirectional associative memory" (which is content addressable with associative memory) in that it can complete fragmented inputs.

Lastly, a very interesting paper by Huyser & Horowitz (1988) demonstrates how, by studying BSB-hypercubes, very considerable purchase is obtained on the critical aspects of conventional neural nets. This paper gives the first real glimpse of the power and advantages of the hypercube approach. In fact, one is left feeling that the work begs the question: "Why having discovered such an illuminating method is it merely used as an aid to understanding a less elucidating but more standard learning machine, namely linear threshold units?" Surely the paper makes a convincing case for hypercube learning machines, *per se*? This point does not seem to have occurred to the authors! The reason for this may be due to the fact that a proper hypercube approach was not actually taken. Rather, the representation used was a degraded 2D equivalent, namely Karnaugh mapping. Yet some of the subtler points of Karnaugh mapping were not even mentioned - conceivably they were not appreciated because problems of sufficient size were not undertaken. If we suspect that a better solution would utilise the more compiled hypercube representation itself then the authors make the telling comment: "An effort needs to be made to invent representations that are more compact than Karnaugh maps ..." Nevertheless, in short this paper is first class.

2.3.4 Other work on hypercubes.

Hypersphere classifiers (Batchelor, 1974), a somewhat unacknowledged (by the P.D.P. school) forerunner of modern competitive learning, are a generalisation of nearest neighbour classifiers which work by measuring the distance of examples in feature space to the nearest example of known class and assigning the test example to that class. The algorithm essentially picks an initial centre of prototypical exemplars called a "locate" and produces the optimal hypersphere with respect to each training example by moving and growing hyperspheres. Its error correction procedure compares the machine and teacher's responses to the decision surface by adjusting the hyperspheres' parameters accordingly.

When the hypersphere's decision surface error rate is sufficiently low the algorithm exits. The major advantages are that the algorithm can out perform Perceptrons, it can accommodate real-valued input and can produce a nonlinear decision surface. Its disadvantages are:

1 Placement of hyperspheres is heuristically based.
2 It may produce degenerate hyperspheres e.g. concentric.
3 It is bottom-up driven: needing to look at all examples it is exemplar limited.
4 It is asymmetrically biased: must be run twice: for positive and negative instances.
5 It is sensitive to noise which is handled statistically.
6 It is only weakly extensible to multiple classes.

Several other references on hypercubes are worth very briefly mentioning. Amit (1989) discusses the important contribution of hypercubes in our understanding of n-dimensional problems. Kandel & Lee (1979) generalise boolean lattices such as hypercubes as a basis for their fuzzy set theory. Amarel (1986) discusses the importance of partially ordered lattices (e.g. hypercubes) as a basis for program synthesis. Youssef and Narahari (1990) in studying communication networks propose the advantages of combining banyans and hypercubes. De Kleer (1986) describes a non-associative hypercube for multiple fault diagnosis and ATMS where the algorithmic essentials are that the hypercube is climbed nodewise. There are a large number of papers on various hardware aspects (e.g. communication) of the many hypercube machines under development and a number of papers on the software routing aspects of these machines. Routing is concerned with communications between nodes. Hypercube routing algorithms are typically depth-first and concerned with matters such as the reliability of the processors sited at each node. An example is Chen & Shin (1990).

2.4 Conclusion.

This chapter was in three parts. Firstly we began with an overview of learning (section 2.1). This literature survey was sufficiently broad in scope to briefly analyse all the major fields of learning (Machine Learning, Knowledge Acquisition, Connectionism and Genetic learning) together in some cases with their major subfields. (The only exception being certain older mathematical and statistical techniques not normally included but often used as a datum against which to compare the performance of learning engines). Dissection and comparison of these machines highlighted their strengths and weaknesses.

Secondly, we considered the more modern approaches to integrated learning systems (section 2.2) which attempt to combine the strengths and eliminate the weaknesses of the above older methods. Our analysis of these machines led to a rejection of their methods for the copious reasons given.

Thirdly, this failure suggested a requirement for a more basic and unifying form of integration. This search led to a representation which may underlie all learning machines and is explicit in those given in section 2.3, namely hypercube learning machines.

In conclusion, it has been the purpose of this chapter to impart not just the facts about the theory and practice of learning machines, or even (obviously better) an analysis and comparison of such machines. Rather, we have struggled to impart to the reader something much more *subtle* - namely **an intuitive feel for the subject.** Hopefully, the reader has *en-route* spotted the various consistent threads in this argument. For this reason the subject was analysed both in scope and depth in order to identify what we feel to be significant and consistent reasons for failure. If we have succeeded in this difficult task then we would hope that the reader may be better able to appreciate why we so strongly strive to discern the essentials of low level learning and why the resulting insight in turn suggests a possible solution is a lattice-theoretic hypercube representation. This alone we feel justifies our otherwise unjustifiable case for the very novel learning machine to be theoretically defined in the next chapter.

CHAPTER 3

The Theory of Hypercube Learning.

3.1 THE SCIENTIFIC METHODOLOGY OF HYPERCUBE LEARNING.

A thesis should tell a developing "story." The work then becomes easier to follow and the whole more comprehensible. We have already outlined the contents of each chapter in chapter 1. It remains to put some detail on the approach taken in order that the unfolding story can be previewed and born in mind as an overview. That is the purpose of the following.

Learning is a very difficult subject. It has been a graveyard for many a research project. The marginal successes in this presently active and important subject lead us to make explicit the scientific approach to learning taken in this project. The difficulties are not so much caused by the complexity of the subject matter, which certainly does not help, but rather by an underlying cause. In contrast to earlier sciences, AI typically involves a very large number of variables. Trying to limit the number of variables is the essence of the above difficulties. It is particularly difficult to notice all the assumptions and variables not accommodated by the theory, in order to obtain meaningful results both theoretically and experimentally. Let us refer to this general problem as *"hidden variables."*

This amounts to a division between two schools of thought. Either one can attempt to build a technology based upon a totally inadequate set of sub-technologies, and hope to learn major principles. The danger in this case is the difficulty of credit and blame assignment with respect to the resulting successes and failures. Or, one can attempt to construct the technologies required for the simplest possible building bricks, which will ultimately be necessary. The problem in this case is that little is *apparently* achieved, of value, for some unknown and possibly considerable period of time. That is, until the technology matures sufficiently to put groups of such bricks together. [1]

We strongly believe that the present trend away from the complexities of symbolic processing towards the simpler non-symbolic processing is a considerable step in the right direction: a necessary step towards limiting the number of hidden variables involved. *Testing this very point is one raison d' etre of this project!* Against these difficulties it is necessary to attend even more rigourously to a scientific approach.

---

1 A third revision may be required at some stage since, for example, it is possible to perform empirical learning on high level information. That is, the above dichotomy may turn out to be the extreme points of a range. Even worse, the concepts within which knowledge is expressed may contain hidden, high level, assumptions. These unrecognised "holes" can be a source of mischief since we should never forget that our idea of the world is just that, an idea!

In science there are experimentalists and theorists. The theorist is unconvinced unless he can mathematically prove or logically reason the general case. The experimentalist is unconvinced unless he can demonstrate the effect experimentally. Rather we need a *balanced* viewpoint. The following expansions of these two viewpoints are specious in that they each present only half of the picture.

The experimentalist says "the proof of the baking is in the eating." (Baking is tested by eating - i.e. the a posteriori is the important part). Behind every theory is an experiment or experiments. Following every theory is a new experiment or experiments to test the theoretical predictions. Only if the theory, born of experiment, leads to successful new experiments is the experimentalist convinced of the theory's predictive value. Theorising is no good unless it works in the real world.

The theorist, in effect, says "the proof of the eating is in the baking." (Eating is tested by baking - i.e. the a priori is the important part). Behind every experiment is a theory implying prediction and expectation. Following every experiment is the old theory retained or a new theory proposed. Only if the experiment, born of theory, leads to a successful new theory is the theorist convinced of the experiment's general value. Experimenting is pointless unless it is precisely coupled with its theory. Science is, of course, a spiral of theory, leading to experiments, leading to new theories, leading to new experiments, etc. However, for simplicity we consider, herein, the elements of only one such loop.

3.1.1 Theory.

Many AI theories fail. The secret behind successful theory requires essentially two things. Firstly, metaphorically, one or two hooks into the real world are required to some crucial and fundamental aspect of the real world, that is, to something all-embracing and always true. This hook may be to something so simple and obvious about the real world that it is very hard to find. In which case it may take considerable scientific intuition to discover and then, further, to realise the significance of such a pivotal fact. The theorist may fail here. In AI, theoretical failures typically reveal themselves only later: in brittleness and inability to scale up to the real world.

Secondly, straight deductions are made based upon this fundamental fact. The result is the theory and its attendant theoretical predictions. *Theorising is of no use if it is based upon poor appreciation of reality and/or is badly deduced.*

Consider an example. Greek science was weak partly because they did no experimental work. They were therefore unaware of important basic facts about the world. Aristotle said that if a force pushes against something the object will stop moving shortly. Newton said if you *experiment* with the same force acting on the same object on different

surfaces, then you will find that when the resistance is less the same force moves the object further. So the real law is *with no resistance it will move forever in a straight line.* That simple observation and resulting theory was the birth of physics intoned in Newton's first law. *Every body continues in its state of rest or in uniform motion in a straight line unless compelled by some external force (resistance) to act otherwise.*

3.1.2 Experiment.

Experiment is as we have already seen, in a sense, the inverse of theory. Many AI experiments fail to scale up to the real world. The secret behind successful experiment requires two things. Firstly, after building the theory into a system, we metaphorically require one or two hooks into precisely a point of maximum leverage in the theory - points most likely to lead to failure.

Secondly, we require careful experimentation such that there can only be **one** possible explanation of the result demonstrated. If the experiment is badly performed, with a weak or incorrect understanding of the theory and/or the conclusion has many possible alternative explanations, then the experiment demonstrates nothing. Many experimentalists fail on this second point due to the number of *hidden variables* involved. Typically, failure of experimental technique reveals itself only later when further experiments produce very different results thereby uncovering new variables. (A recent well publicised example being "cold fusion." In AI, an excellent example is the still celebrated "blocks-world" Ph.D. thesis of Winograd (Winston, 1984) - by Winograd's own, later, very thorough admission, (Winograd and Flores, 1986). The result is the experiments and their attendant experimental results. *Experimenting is of no use if it is based upon poor appreciation of its equivalent theory and/or is badly performed.*

Consider an example. Yuri Gagarin, the acclaimed Soviet astronaut, upon completion of his first orbit around the earth, sent back a triumphal message to earth. In effect he said: "I've had a good look round up here, there is definitely no such thing as God!" Clearly the theory, in this case, that God lives in the heavens which were his own creation, was primitive in the extreme. Leaving that point alone, suppose that Yuri looked in 500 places above the Earth, does that number of experiments prove the theory? Of course not. There are rather a lot of other places in the Universe also to search! Indeed, the universe is so large that Yuri could have carried on looking for numerous lifetimes and yet the number of experiments performed as a proportion of the possible whole would still be so infinitesimally small that nothing would be proven.

This situation is very typically the case with experimental AI work. For all non-trivial theories, no matter how many experiments are performed, that number, as a percentage

of the total data space of possible experiments to test the theory, is so infinitesimally small that nothing is proven. To attempt to do so is to grope in the dark. Nothing is achieved. The data space is so large that the results obtained may well be atypical. Any given experiment only tests one point in the data space. **Generality cannot be *experimentally* demonstrated.** Hence we conclude the following. *For all non-trivial theories, no matter how many experiments are performed, it is never possible to prove the theory.*

It is, however, possible to *disprove* the theory by experiment. Indeed one carefully designed experiment will elegantly achieve this. The correct experimental procedure is, therefore, clearly and most efficiently the second approach. The correct experimental approach is, therefore, to attempt to demonstrate that the theory is *incorrect.* All else is futile.

3.1.3 The scientific cycle for machine learning.

We summarise the above discussion with reference to machine learning as one loop in a cycle.

1) We require solid links into experiments on reality concerning some fundamental, eternal fact about learning.

2) Correct theoretical deductions based upon these links lead to:
the theoretical predictions.

3) The theory is built, as closely as possible, into a machine learning program so that the program behaviour *is* the theoretical predictions. Solid links into the theory are sought with a view to well designed and carefully performed experiments on this program focusing upon some point of possible failure in the theory.

4) Experimental purchase applied at this failure point, to carefully exclude extraneous explanations, upon analysis leads to:
the experimental result,
which may then *disprove, but never prove,* the theory.

Should the theory *not* be shown to be false, then the theory is not proven, merely retained pending the next experiment, rather than discarded, as in the case of it being disproved. Clearly we ought now to fill-in for the details of the points 1 to 4, above, for our target the hypercube learning machine. However, the intention, as above, was to set the scene for the ensuing chapters. We will not spoil the "plot" for the reader. The details of the hypercube cycle will be left until the conclusion of the thesis.

The above cycle was precisely followed chronologically, but to ease the reader's task we first provide a basic introduction to hypercubes before considering points 1 and 2.

3.2 INTRODUCTORY CONCEPTS.

The stated aim of this project (section 1.3) was to build a coalesced learning engine. This is becoming quite a trendy thing to do. For example, Ellman (1989) states his interest in building a machine to integrate EBL (section 2.2.3) and empirical learning in order to overcome the shortcomings of each. Gangly and others are interested in building a machine to integrate back propagation and EBL (Gangly, 1987).

3.2.1 Integration and the ad hoc.

Some of the dangers of integration were pointed out in section 2.2.3.5. The biggest danger with any integrated machine lies in the possibility of an ad hoc juxtaposition of previous work. Nevertheless, an ad hoc machine may work quite well in practice. For example, Quinlan transformed ID3 by ensuring reasonable noise immunity via a Chi Squared test of the data (Quinlan, 1988). Chi Squared has nothing to do with decision trees, the information theoretics of ID3 and its associated windowing mechanism and so we label the resulting construction as ad hoc. Yet, in practice, this enlarged (ad hoc) ID3 is quite impressive in performance. So what is wrong with an ad hoc solution?

The world record for the longest held theory is 1400 years - held by Ptolemy's "wheels within wheels" theory of planetary motion and the heavens. The deficiency with this theory was that it required seven machineries (as seen in a working model built later) not one. Eventually, the work of Copernicus, Kepler and Galileo resulted in one theory which also then explained certain minor aberrations such as the anomalous behaviour of Mars, in apparently "looping the loop", as seen from the Earth.

Thus the real objection to an ad hoc assemblage of methods is that it is intellectually unsatisfactory and may well thereby fall in the course of time. We really require **one** unified theory and approach, not **many**. Hence we take the view that one way to avoid the ad hoc quagmire is to search for common ground or a lowest common denominator, as the starting point. A reasonable basis in the search for common ground amongst the three major approaches to learning is to be found in the hypercube representational work of the authors cited in the previous chapter. In this respect, the work of Anderson and others on BSB (section 2.3.3) in Connectionism, the work of Ganascia in Machine Learning (section 2.3.2) and Oosthuizen in Knowledge Acquisition (section 2.3.1) are all of particular interest.

3.2.2 Representation and the Hilbert hypercube.

Why start with a representational aid? We recall Minsky's sound advice (section 1.1.3): "learning is a non-problem, the real problem is **representation**." Indeed, most of

the work in AI throughout the 1970's concerned the discovery of the crucial importance of a flexible representation (section 1.1.5). How is it possible to obtain a totally flexible representation? Again the solution is to find the lowest common denominator. In effect, von Neumann has already solved this problem. The answer was a binary machine the microcode of which forms the basis of all present-day serial machines and we all know just how flexible this machine has proved to be - as predicted by the Church-Turing thesis.

We have here, however, made light of the importance of later work which prescribed how to combine binary into larger grain sizes using assemblers, high level languages, operating systems, etc. This present work again takes its cue from the work of Anderson, Ganascia, etc. in utilizing a **binary hypercube representation** or **Hilbert hypercube**. However, by analogy with the above early work in computing, the problem of how to address larger grain sizes (with some analogy of an assembler etc.) remains largely unspecified in this present work since this accommodation can take place *outside the basic machine* - as indeed is the case with conventional computers. It follows that, what is as purely binary to the hypercube machine can independently take *any meaning or form as viewed externally*. Yet the problem of knowledge is of crucial importance to machine intelligence.

3.2.3 Domain specific knowledge.

There is a certain asymmetry in our approach towards computer science in general and Artificial Intelligence in particular, in that, on the one hand we ask the machine to react intelligently with respect to the world and users, and on the other hand we provide it with such a paucity of contact with the real world that we find the former requirement inevitably leads to our description of the machine as an idiot savant. The problem of lack of knowledge of the real world in such systems is all pervasive and it is now realised, for all non-trivial systems, inescapable. Lack of such knowledge leads to a ceiling of unreliability which induces problems such as brittleness in expert systems. Yet, knowledge acquisition is acknowledged to be a time consuming process. The general problem is known as the knowledge acquisition bottleneck (Feigenbaum et al., 1988).

Early work on expert systems, in particular the DENDRAL project by Buchanan *et al.* (Buchanan and Feigenbaum, 1978), identified knowledge as the crucial component in Artificial Intelligence systems. The essential requirement was found to be the need for vast amounts of domain specific knowledge. The earliest attempt to build a knowledge acquisition system was by Davis (1980) in the mid-seventies and known as TEIRESIAS. Today nearly one thousand papers exist on knowledge acquisition, which is a testimony to its importance. Furthermore, associated subjects such as machine learning and neural nets can be seen as making oblique attacks on certain aspects of the knowledge acquisition problem.

Hence a second great tenet of AI is Feigenbaum's advice that (paraphrasing) "*The key to AI is masses of* ***domain specific knowledge*** *and a small reasoner, as opposed to its inverse.*" The hypercube representation has the advantage that it acts as a vast memory reservoir ready to be filled with the domain specific knowledge which then finds its unique place within the hypercube in a **totally structured and orthogonal manner**. This has considerable implications as we shall see later.

3.2.4 Spatial representation of the hypercube.

The spatial representation of the general boolean quantity makes visualization easier. If A is a general boolean quantity with n components each taking one of two values then there are 2^n cases. These cases can be represented in the corresponding n-dimensional space. We require, firstly, that the axes of each of these dimensions are mutually perpendicular to all the other n-1 axes. Secondly, the coordinate values along each of these axes are restricted to just the two simple boolean values, namely 0 and 1.

Figure 3.1 (a) illustrates a one dimensional hypercube H^1. This is just simply a line segment with a node at either end. Figure 3.1 (b) illustrates the two dimensional hypercube H^2. The result is a square with a node at each corner. Figure 3.1 (c) illustrates the three dimensional hypercube H^3. The result being a cube with a node at each vertex. A hypercube is therefore an orthogonal network of arcs, either based on mutually perpendicular axes or else parallel to these axes, in the form of straight lines joining at vertices or nodes.

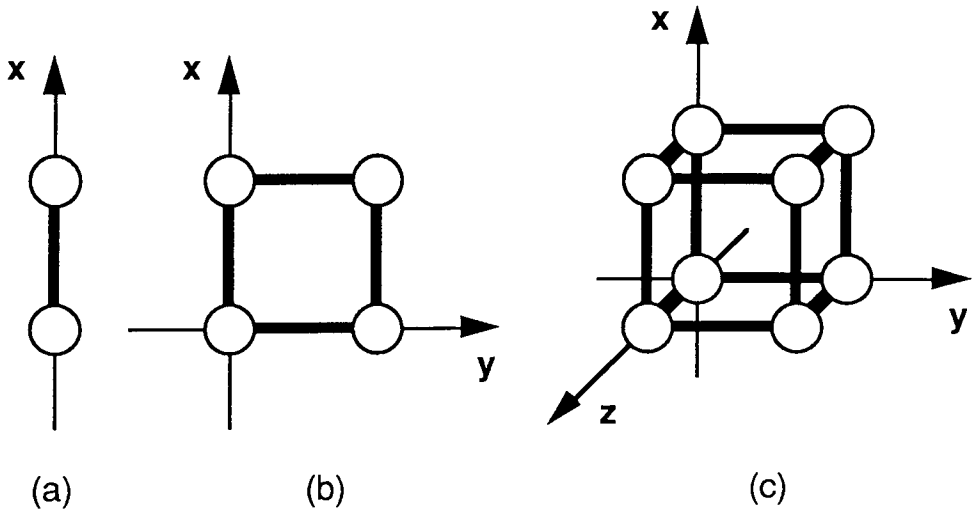


Figure 3.1. H^1 to H^3 with axes.

If the axes are dispensed with then the equivalent hypercubes for H^1 to H^3 inclusive can be represented with coordinates as Figure 3.2.

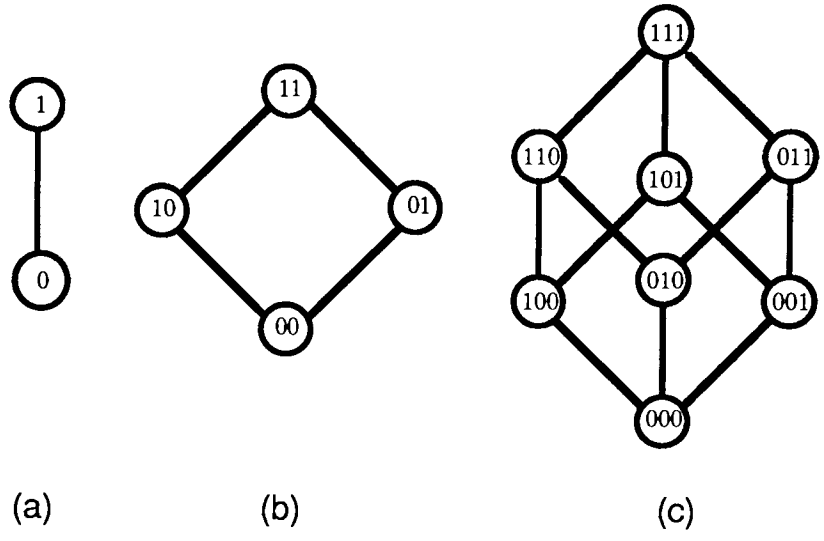


Figure 3.2. H^1 to H^3 minus axes, plus coordinates.

Representation beyond n = 3 presents difficulties. The square (n = 2) can be correctly represented on a two dimensional surface such as paper. The cube (n = 3) already begins to present difficulties for two dimensional representation. We cannot draw all three axes perpendicular to each other as required above.

Nevertheless, since we appear to live in a three dimensional Euclidean world, a good impression of a perspective viewpoint of a cube in three dimensions is easily obtained. By pointing out the deficiencies of this representation of the cube it is easier to understand the next dimension - the fourth dimension, as Figure 3.3.

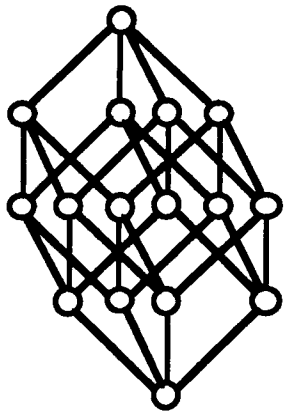


Figure 3.3 The fourth dimensional hypercube H^4.

In order to make certain following points clear we take the liberty of redrawing Figure 3.3 in a less abstract and network form as the more easily visualizable representation in Figure 3.4. The diagram now emphasizes certain arcs by shading in order to make its perspective obvious and we have disregarded any specific representation of the nodes in order not to distract from the effect (2 boxes connected by 6 boxes, one per mutual face).

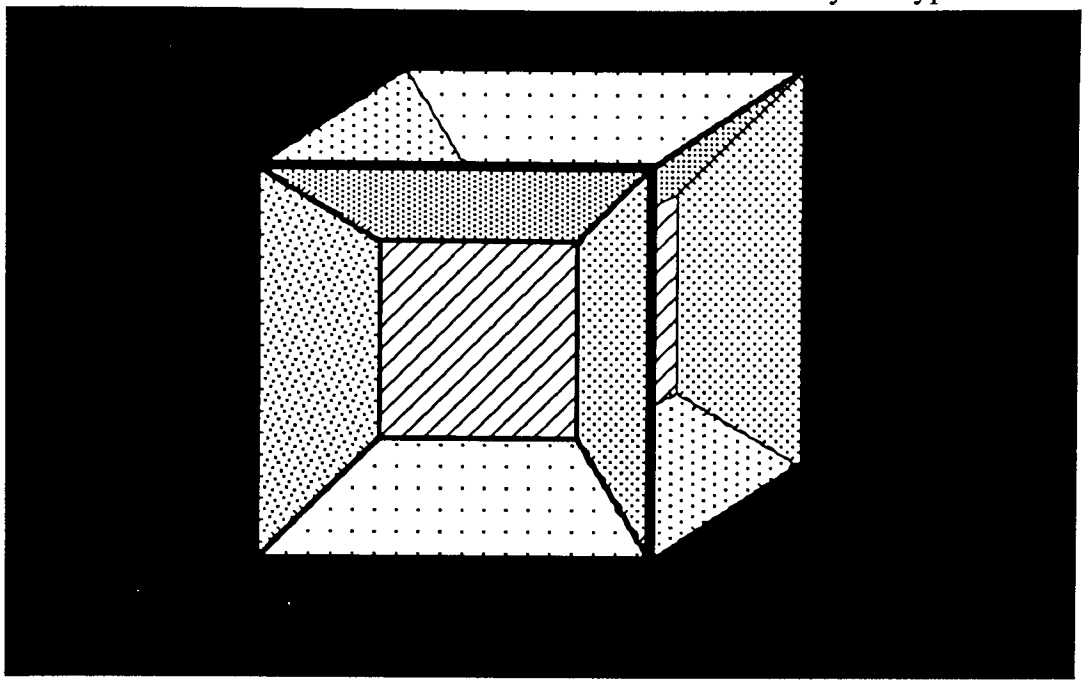


Figure 3.4. Emphasizing the fourth dimensionality of the hypercube H^4.

If the reader now attempts to 'see' the inner cube within Figure 3.4 as if it were a model in 3D then once again, as with the cube with respect to 2D, we see a further dimension which can only inadequately be represented. It is inadequate because, as before for H^3, this final dimension is not mutually perpendicular to the other three dimensions in our representation. Nevertheless, by analogy with H^3, due to the orthogonality of the hypercube, it is clearly a reasonable attempt to picture the fourth dimension.

The hypercubes beyond the second dimension therefore become increasingly difficult to represent. This does not detract from the fact that this Euclidean geometrical interpretation of the higher dimensional hypercubes remains convenient and it is even possible to partially appreciate and visualize such structures with practice. Mathematically, the alternative to a geometrical interpretation is an algebraic description. The author has found on working with higher dimensional spaces that they are most easily accessible in the first instance geometrically and then in more detail by translation into algebra. Although this practice may not suit every reader, we follow this procedure in our descriptions within this thesis, since we then have two perspectives. [2]

An example of the convenience of the geometrical interpretation is our **double and join rule** for the general hypercube. That is, the rule by which to create H^{n+1}, given H^n for any dimension 'n'. The rule, see Figure 3.5, works as follows.

---

2 One's preferential form of description depends upon what is personally found to be the most easily understood form of description. For example in describing how to get from A to B, one may immediately get pencil and paper and draw a diagram. Another may give the description purely in words. A heavily biased form of communication given in a less easily assimilated form for a particular reader or listener can make opaque that which is found to be near trivial in an alternative form of communication.

Given H^n:

1) double it - make a copy alongside the original, and

2) join it - join corresponding nodes in the original to those in the copy.

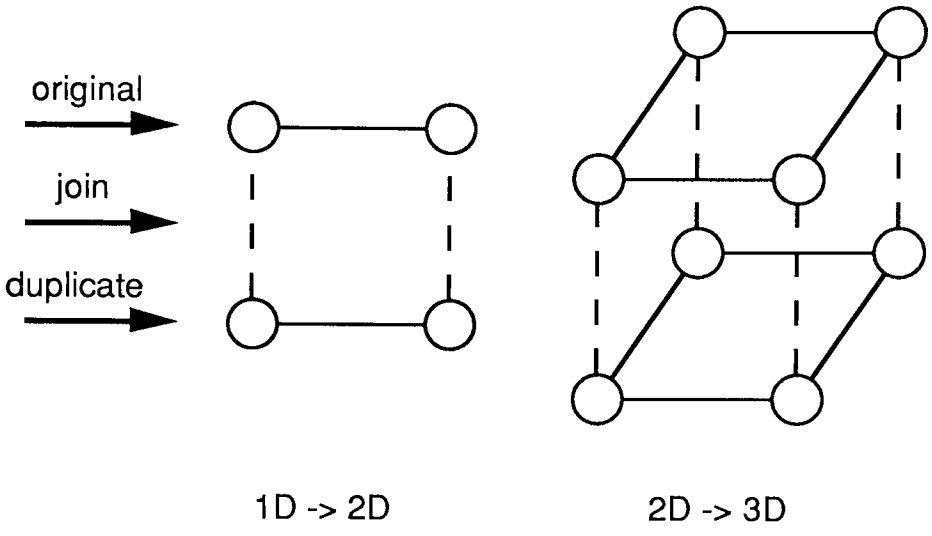


Figure 3.5. Illustrating the double and join rule.

From now on it will be convenient to to use the term hypercube for all values of n in the term H^n. Any given hypercube can be drawn as 'seen' from an infinity of perspectives. For the fourth dimension, two of these possibilities appear to be topologically distinct - making the copy on the apparent outside or inside of the original and then joining both along this newly defined dimension. This is incorrect. The same hypercube results, even if they appear to be very different structures, indeed irreconcilable structures. The problem lies in our inability to fully appreciate the fourth dimension, as we discovered by making deformable models of both cases and twisting the one network into the other, see Figures 3.6 and 3.7.

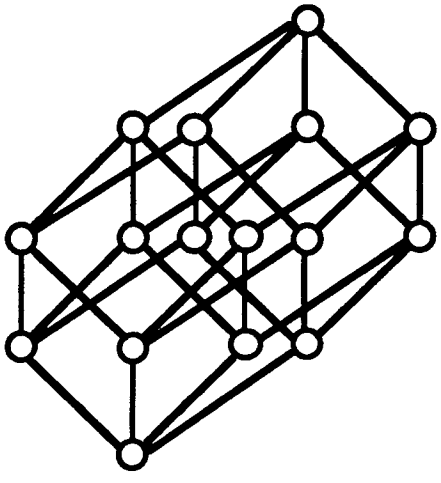


Figure 3.6. H^4 by outside (alongside) duplication.

The fourth dimensional hypercube is known and found to be a useful concept in several other diverse subject areas outside computing such as electronic circuit design, cryptography, mathematics and psychology where it is called Necker's cube. The fourth dimensional hypercube in Figure 3.7 is used in special relativity, in Minkowski's fourth-dimensional space time, and called the "Tesseract" (Rucker, 1977). Higher dimensional hypercube concepts have also been entertained at various times, although they are not given separate names. For example, tenth-dimensional hypercubes have been used in General Relativity (Misner et al., 1973). [3]

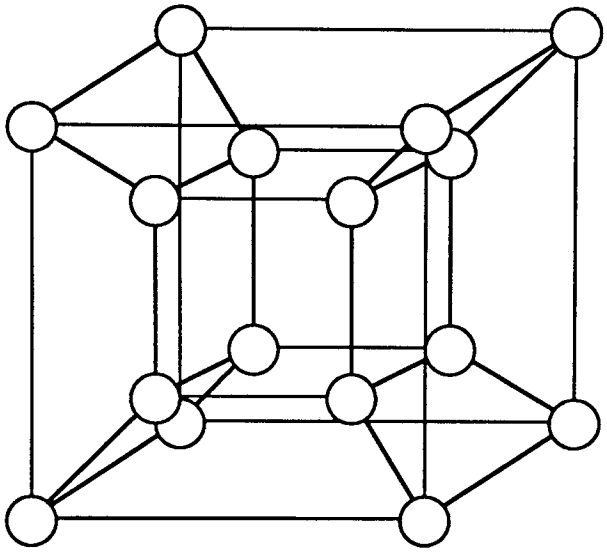


Figure 3.7. H^4 by inside (perspective) duplication.

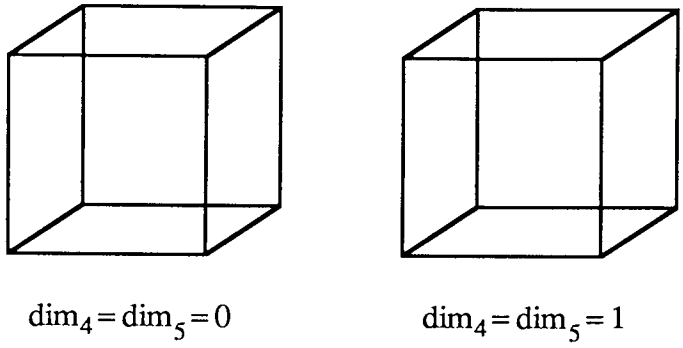
Notice that in the case of H^4 in Figure 3.6 there are 8 distinct but overlapping cubes forming the hypercube. These 8 cubes are much easier to see in Figure 3.7. It is a useful exercise for the reader to verify this in both cases since from here on the reader will be required to get used to such visualisation. In the fourth dimension there will indeed be 8 perfect cubes, although only two are even apparently cubic in Figure 3.7. The distortion in the remaining 6 is again due to the inadequate representation of H^4. Similarly other perspectives introduce other distortions. For example, the diamond shapes of Figures 3.1 (c), 3.3, 3.5, and 3.6. The main point, however, is that we observe 4 lines in the second dimension, 6 faces in the third dimension, 8 cubes in the fourth dimension, etc. This observation is important later as one component of the learning algorithm. We generalise it as our **double rule**.

The double rule states that given H^n there will be 2n distinct H^{n-1} first level subhypercubes.

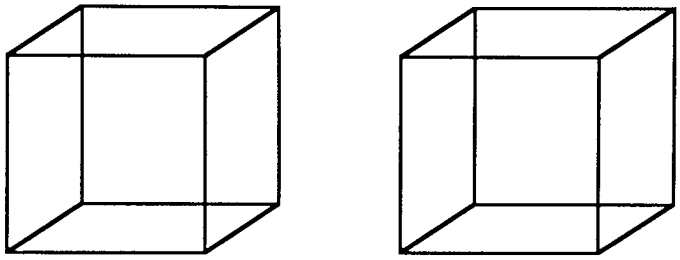
---

3 The following observation appears to be most interesting. Unfortunately we are at a loss as to how to capitalise on this insight. Space can be represented in the mathematics of hyperspace and to a first approximation by hypercubes, as above. Mind, it seems, can only appreciate the external world by itself re-representing the external world and perhaps in ways which are equivalent to the same hyperspace formalism.

The 'double' part of the "double and join rule" can be reobtained by considering the boolean dichotomy along any dimension. For example consider H^4 as in Figure 3.6. If the coordinate value along some particular dimension say $dim_i = 1$ then corresponding nodes form a cube whose nodes are distinct at every vertex from that cube obtained for $dim_i = 0$. It follows that for H^5 and for i = 4, 5 we obtain the four distinct cubes as in Figure 3.8.



$dim_4 = dim_5 = 0$ $dim_4 = dim_5 = 1$



$dim_4 = 0, dim_5 = 1$ $dim_4 = 1, dim_5 = 0$

Figure 3.8. The four distinct cubes in H^5.

These four cubes can be made to stand out in H5 as seen in Figure 3.9.

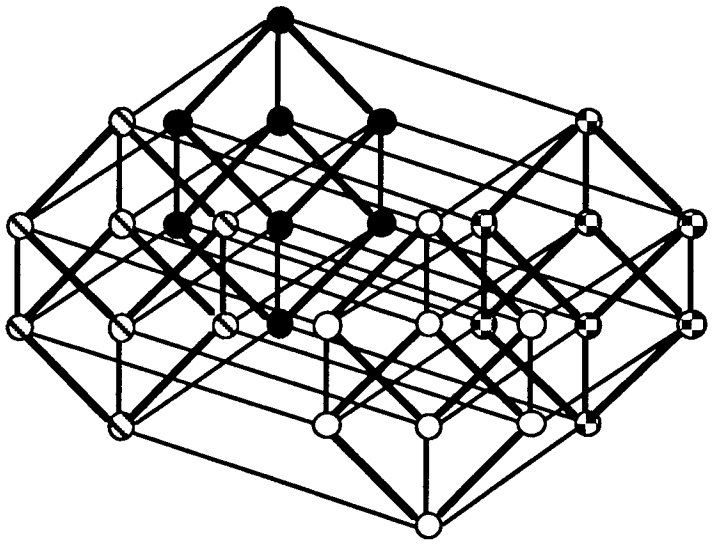


Figure 3.9. The 4 cubes of H3 in H5.

3.2.5 Invariance and variations.

The problem of invariance is fundamental to human information processing. We are able to recognize an alphabetic letter such as a 'T' irrespective of its position, size, rotation, colour, contrast, texture, whether black on white or white on black, font, etc. We recognize particular words even though the component frequencies, amplitudes, or time durations of such sounds, between the same or different people, are subject to considerable variations. The attainment of invariance capable of dealing with such difficulties is clearly a worthwhile goal, but how is it to be achieved? The solution may lie, as indeed is strongly suggested by neurophysiology experiments, in forming a representation sufficient to enable capture of the full dimensionality of the problem. We present a simplified example using a hypercube. Consider an example in H^4 representing the components of the letter 'T', as diagramatically illustrated and ignoring arcs, in Figure 3.10, together with associated reflections and rotations as indicated.

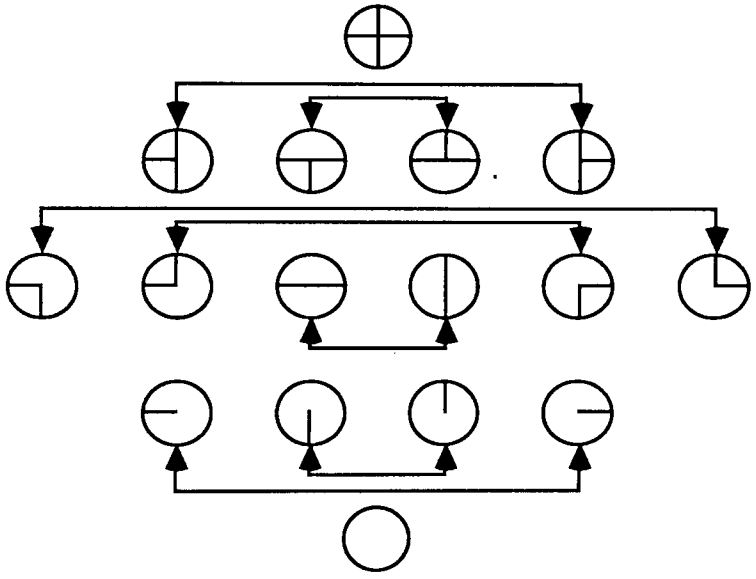


Figure 3.10 'T' representation in H^4.

At the base of the hypercube, we see the null set. On the second level, the components of a simple four quadrant 'T' are illustrated as the dimensions of a four dimensional hypercube. All possible variations on these basic dimensions are seen in the higher levels. On the fourth level, all possible invariance mappings of the 'T' within the terms of reference of the problem are clearly seen along this level. That is, 'T' rotated about the four quadrants. The lattice is therefore seen to capture all possible invariances and variations within the scope of the problem. A related approach to the problem of invariance was suggested by Hinton as discussed in Rumelhart and McClelland (1986a). In Hinton's representation there are two sets of features plus a mapper. Hinton's machine is, in fact, a hypercube equivalent disguised in the form of a coordinate mapper.

3.2.6 Neurophysiological evidence.

Our brains contain of the order of 10^{11} neurons and 10^{15} synaptic connections. Yet large areas of the cortex appear to be concerned, possibly exclusively, with the subconscious, for example, the visual "striate" cortex and its associated visual cortical areas, such as "v2", "v3", "v4", "v5" etc. On the other hand, our understanding of the workings of even a single neuron is at present incomplete. Whitfield (1984) observes that we are unaware of what it is that one neuron learns from its predecessors. In other words, the coding scheme used, the "semantics" (to a neuron!) of the information flow, etc. are all unknown to science. Whitfield further states that it is unknown whether it is even meaningful to pose such questions. Other neurophysiologists go much further and question even the validity of the computer paradigm of the brain. For example, Whitfield suggests a possibly different paradigm by the phrase that "it is not so much the wiring as the chemistry of the wiring that matters", and he goes further by suggesting that we are dealing with a chemical machine, like some vast chemical works, of completely unknown origin.

In the face of such difficulties, to attempt to model the possibly billions of neurons involved in the higher level learning processes of the brain seems, to say the least, ambitious. The possible number of brain states - calculated from *very* simplifying assumptions *ignoring* altogether synapses and neurotransmitter states (Iversen, 1979) - approximates the number of elementary particles in the known universe (Whitfield, 1984). Perhaps, therefore, it is not surprising that AI machines have often failed to reach commercial acceptability. Furthermore, in the few areas of success such as expert systems the machine is still characterised by minuscule domains of application and unpredictable brittleness. One possible reason for this state of affairs may be the lack of breadth of knowledge which might be obtainable from the stronger foundations of a low level learning system.

To be *really* low level we should not even assume that the data involves knowledge or concepts (Stonier, 1986), that the data has some predefined representation to be uncovered, or even that we are dealing with symbols! Do we really know, or can we even reasonably *assume*, that we *necessarily* must use at some sub-conscious level, concepts and symbols? Consider for example, the extent to which we are aware of our earlier stages of visual, tactile or audible object recognition. Hence we may be best to assume *nothing* but the *multidimensionality of the input data* and turn our attention to the immediate problems of constructing a very low level *data* acquisition machine. This is the approach we have taken.

What, is known about the brain is intriguing and exciting. The structure of the representation appears to be very important. Experimental investigations by neurophysiologists has convincingly demonstrated that the brain is highly multidimensionally structured (Ballard, 1986; Hubel et al., 1978; Hubel & Wiesel, 1979). The results from microelectrode penetration of the cerebral cortex has provided convincing evidence that **the**

brain is exceptionally highly structured despite its appearance otherwise in golgi stains (Hubel, 1988; Rose and Dobson, 1985). A microelectrode pushed slowly in a straight line in the visual parts of the cortex suggests that the active surface layer is divided into areas of about $2mm^3$ known as "hypercolumns" or "orientation columns." Hypercolumns (of 10^4 neurons) appear to be identifiably independent processing units. Each half of the hypercolumn takes its input from one of the eyes. Work by Hubel and Wiesel, Zeki and others has identified highly structured and uniform subunits within the hypercolumn (Hubel & Wiesel, 1979; Blakemore, 1990). Typical work done to date has illustrated specialised hyperfields within hypercolumns concerned with aspects of lines of varying thickness, orientation, colour, movement etc. All modalities appear to operate by similar principles. Therefore it is sufficient to consider vision. Retinal processing and various relay stages on the way up to the visual cortex mean that the hypercolumns at the visual cortex can work with larger visual fields such as bars and lines. A glimpse at our understanding of a typical hypercolumn is provided in Figure 3.11.

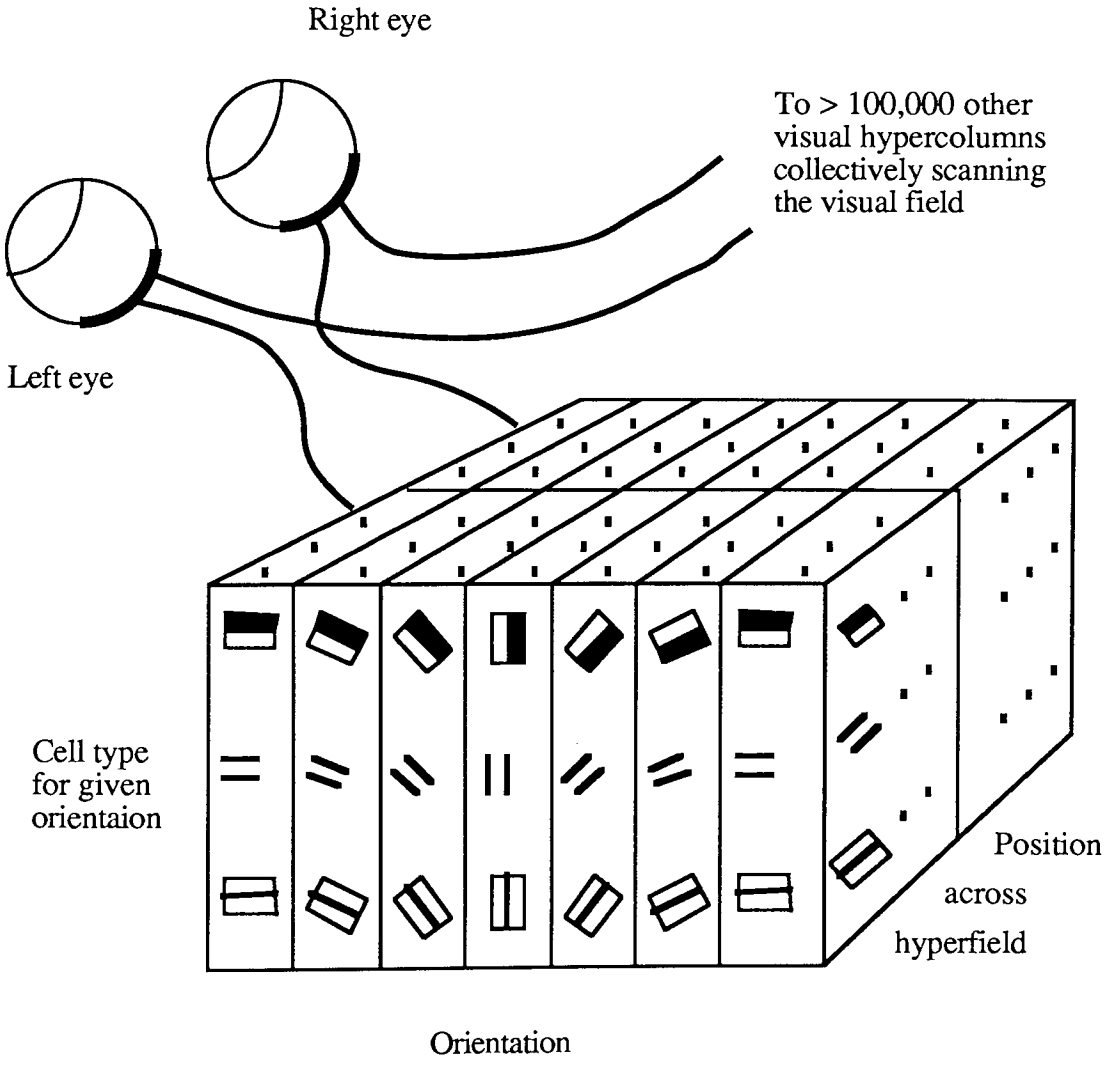


Figure 3.11 Neural hypercolumn : specialisation = bar orientation.

Across the various dimensions of the hypercolumn lines of slowly varying types are evident. For example, 180 degrees is catered for by 18 lines at uniformly intermediate intervals. Lines vary from thin to thick, from bars to parallel lines etc. with again about 18 types. Visual field position is the third dimension. The uniformity of structure is impressive. The basis for each hypercolumn could be summarised as an **approximate method of obtaining all possible variations of any particular problem** considered. This is **hypercube-like** (compare Figure 3.10). The same organisation seems to exist throughout the brain (Hubel, 1988), for example, "grandmother cell" (Amit, 1989) hypercolumns have also been reported (Blakemore, 1990). It may be "brute-force", but clearly it works. [4]

3.2.7 Structured inductive knowledge acquisition.

The problem of knowledge acquisition is very difficult because it is multifaceted. The first difficulty is the sheer volume of knowledge required. Human memory capacity seems to be of the order of 10^{15} bits. The real problem, however, is that human memory is so integrated (only 7 or less neurons interconnect any 2 neurons in the brain) and knowledge so readily assimilated and retrieved that the temptation is to conclude that "You cannot *program* A.I.." Very fast, highly parallel, automatic, *learning* systems seem to be required to handle such complexity, for example, inductive machines which learn by example such as neural nets or the ID3 (Quinlan,1983) series of machines. A second difficulty is the need for *structure*, since an amorphous system, such as the first generation rule based expert systems, would be quickly overwhelmed by the complexity aspect. Steels (1986) suggests the need for a deep-model learning system. Similarly, Jackson (1986) warns against multiple paradigm, unstructured ad hoc AI systems. Again, there is a pressing need to make neural nets more structured (Falhman, 1988) since unstructured nets lead to severe problems of opaqueness when they fail or get "stuck." The advantages of structure were also underlined by Shapiro (1987) by his work on structured inductive knowledge acquisition using an ID3 machine.

The Nobel prizewinning work of Hubel and Weisel on the structure of the retina and associated cortical layers is also very relevant (Hubel & Weisel , 1979). As we have just seen (in section 3.2.6) the basic architecture of many areas of the cortex appears to be a surprisingly orthogonal and highly organized structure in the form of three dimensional maps of the sensory input, such that each modality is exhaustively and uniquely situated, as illustrated by Frisby (1979). Hence there is a case for a study of an inductive hypercube machine that is based upon the simplest possible orthogonal data structure, namely the Hilbert

---

4 In this regard, von Neumann's mathematical analysis of the brain concluded that the essence of the brain is *complexity handling* within a "mixed" digital and analog, binary and statistical system thereby trading very much reduced arithmetical precision for much increased logical reliability - this is in accordance with modern neurophysiology, and at variance with much modern neural net research with its six plus significant figure precision - (Neumann, 1958). If we are sufficiently biased with preconceived ideas of what constitutes intelligence to rule out research into "brute-force" reducing algorithms when biological intelligence may have taken this very pathway then we could well be in danger of denying our own intelligence!

hypercube in a CDML (Appendix 2) as a connectionist machine where information is stored in connections rather than memory cells. That is, a cube in n-dimensional space, hence hypercube, where n is the number of individuals contained in the universal set.

Clearly the brain is *not* a mass of hypercube machines. But on the other hand we feel that there is some similarity **at the most abstract level** to the hypercube we discuss herein. This is not a coincidence. Rather, being aware of the experimental work concerning this impressive neural structuring we attempted to find some similar abstract expression of these ideas by extending the current work in learning in the manner outlined. Being a) an abstraction and b) a first approximation we hope to avoid the work being grossly invalidated by latter neurophysiological revelations on the finer details. The result has been the hypercube approach with the many useful advantages such as invariance and variations (section 3.2.5). The simplest possible multidimensional structure is a boolean hypercube representation. This leads to an inductive hypercube machine. Space does not permit a detailed review of all the work on the hypercolumn. The interested reader is initially referred to the references given (Blakemore, 1990; Hubel, 1988; Rose and Dobson, 1985; Frisby, 1979; Hubel *et al.*, 1978).

3.3 THE GENERAL PROBLEM.

Let us argue our way to the heart of the matter. With regard to some given problem we assume that we have set up the equivalent hypercube and that the information obtained specifies the hypercube connections. How is the machine to process this information? In an abstract sense we see a very simple example of this situation in Figure 3.12. This is a fifth-dimensional hypercube formed by the intersection of two fourth-dimensional "tesseracts." In the general case only some nodes are specified. The hypercube is therefore sparse, yet we require the machine to be able to predict the result for any other node! Looking at Figure 3.12 it appears to be quite impossible to predict the state of the remaining nodes. Clearly there are very many possible combinations and no apparent guide-lines by which the machine might choose amongst apparently equal possibilities. Yet this problem is only hard because we have posed it **abstractly**. Analogously, if the problem is posed concretely its solution becomes trivial. Indeed we solve the problem continuously! For example, consider the problem of recognising a tree. As a visual pattern a tree is very complex. Obviously we have only ever seen a minute proportion of all possible tree patterns. Yet, faced with another example of a tree we have little difficulty in recognising it as a tree.

The typical tree would only occupy a tiny proportion of the nodes of that hypercube of equivalent pixel dimensionality. That is, the hypercube would be very sparse indeed.

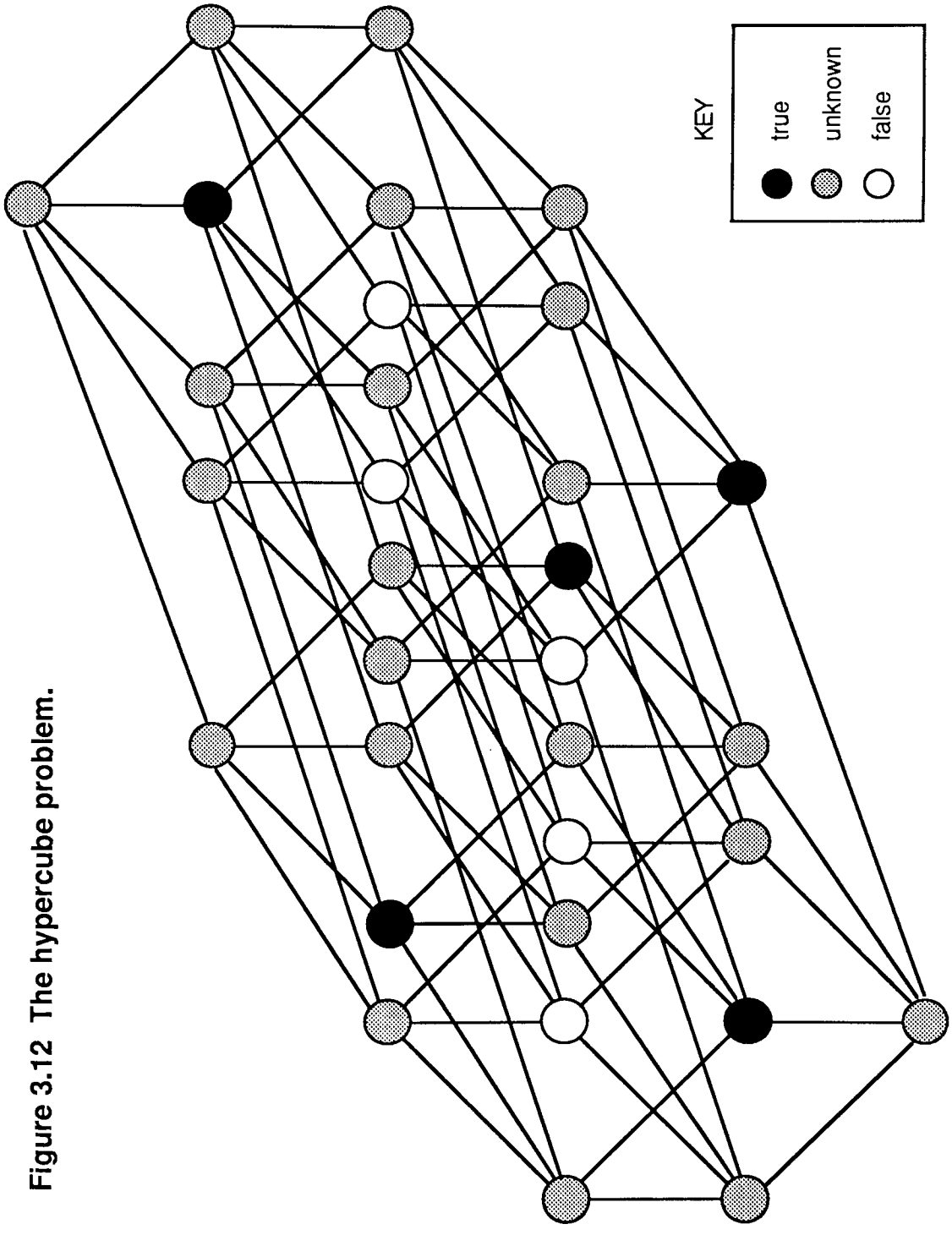


Figure 3.12 The hypercube problem.

Even all known exemplars would only occupy a tiny proportion of all the possible tree exemplars to be found in the generalisation of the concept tree. Yet we find the tree recognition problem trivial. This suggests that the brain uses some very powerful methods to constrain the inherent search.

Gestalt psychologists have proposed that the brain uses perceptual groups to constrain this search. Surprisingly and possibly suspiciously, there is no experimental evidence and no theoretical proofs given as a basis for these proposals. Rather, in order to become convinced, we are asked to look at perceptual groups *phenomenologically*, which can be roughly translated as the following. "Try it for yourself and you will see that it is obviously true." (The reader may smile when remembering that practically *all* of the great breakthroughs in science have come precisely at the point of departure from this *naive* approach - or explanation by use of the "obvious"!!). Disappointingly, (or perhaps we should suggest predictably), no simple neurophysiological basis has been found for these groups - the conclusion being that the concept is still at far too high a level (Roth and Frisby, 1986; Lowe, 1985; Lowe 1987).

3.3.1 Principle X.

We therefore propose that these perceptual groups may themselves be the result of some even more fundamental organising principle. Let us call it **principle 'X'**. Secondly, we consider the possibility that if principle 'X' can be discovered it should then be possible to constructively apply it to good effect, at least to low level concepts, independently of the modality, context, subject matter, etc. even to the extent of whether the nodes depict symbolic or non-symbolic information. [5]

If principle 'X' can be discovered and successfully applied in a number of different problem areas using the hypercube, then principle 'X' would then function as a basis for resolving the **bias** problem posed by Figure 3.12, namely, by what guide-lines do we choose between apparently equal possibilities? If we assume that each of these possibilities results from some other corresponding principle then the space of possibilities delimits the space of possible machines. Clearly, in the general case, the space of possible machines is effectively infinite since it is possible to have an effectively infinitely dimensioned hypercube with an even more nearly infinite set of possibilities.

We take the somewhat disappointing view that out of these infinity of possible machines it only makes sense to try to build <u>one</u> of them! That machine is the machine which

---

5 Yet in another sense, perhaps this is not so unreasonable. It could be argued that we appear to have only one connective structure, namely neurons, which all appear to work in the same way via associated inter-neuron synapses for processing and storing *all* sorts of information: visual, verbal, auditory, etc. One cannot open up the brain and find some other structure doing the work in some other way.

works by principle 'X'. The reason for this huge limitation is that if we build machines which result in different conclusions from our own then we shall have considerable difficulty in understanding them. Indeed we would probably pronounce their conclusions as "wrong" even if logically this makes no sense since logically they would specify an equally valid perspective on the world.

3.3.2 Towards principle X - a worked example.

In order to attempt to discover a good contender for principle 'X' some psychological experiments were undertaken. Figure 3.12 is, as stated, far too hard for us to solve because it is so abstract, thereby missing the usual constraints allowing easy solution. We therefore used a very simple hypercube namely the three dimensional cube as Figure 3.13.

Figure 3.13 (a) poses a problem. We imagine that this simple hypercube encapsulates some concept such that positive examples of the concept are depicted as "true" (black) nodes and negative instances of the concept are depicted as "false" (white) nodes and that the remaining (shaded) nodes are as yet unknown. Note that the problem is still posed abstractly since no actual domain details of the instances are assumed to be known. The question, therefore, is what state should the unknown nodes take? [6]

Figures 3.13 (b) through to Figure 3.13 (e) depict four possibilities and we can attempt to reason a case for each of these. In Figure 3.13 (b), node 001 becomes true and node 100 becomes false. We can reason the case abstractly on the basis of a bias towards weight of evidence. For example, node 100 is connected to two negative exemplars (false) and one positive exemplar (true). The weight of evidence for the node 100 constrains it to be true assuming that we are required to make a binary choice for the node. The decision follows from the *asymmetry*. A similar argument applies to the node 001 constraining it to become true. These constraints are *local*. Hence there is an **asymmetrical bias towards weight of *local* evidence.**

A few further points are in order. Firstly, local evidence is well known to be *unreliable* as a predictor, Boden (1977). Secondly, in the general case for an n-dimensional hypercube it will often be necessary to consider the next-but-one nearest neighbours (and possibly to even further depth) in order to make a decision. Clearly there are problems here.

---

6 The reader may protest: "That depends upon what the nodes depict." Beware! The *obvious* can very effectively mask allegory. A deeper appraisal shows that we are in difficult waters. Let the reader bear with us. The obvious answer turns out to be facile. We will shortly require many pages to separate these waters. An analogy (which should not be taken too far) may help. The child may protest that $y = 2x + 1$ is unsolvable because the result (value of y) is unknown and depends upon a particular example (value of x). The higher student of mathematics sees immediately that it is the equation of a straight line whose slope and intercept are apparent. All is known from abstract consideration without us ever needing to know what x and y mean in the particular case.

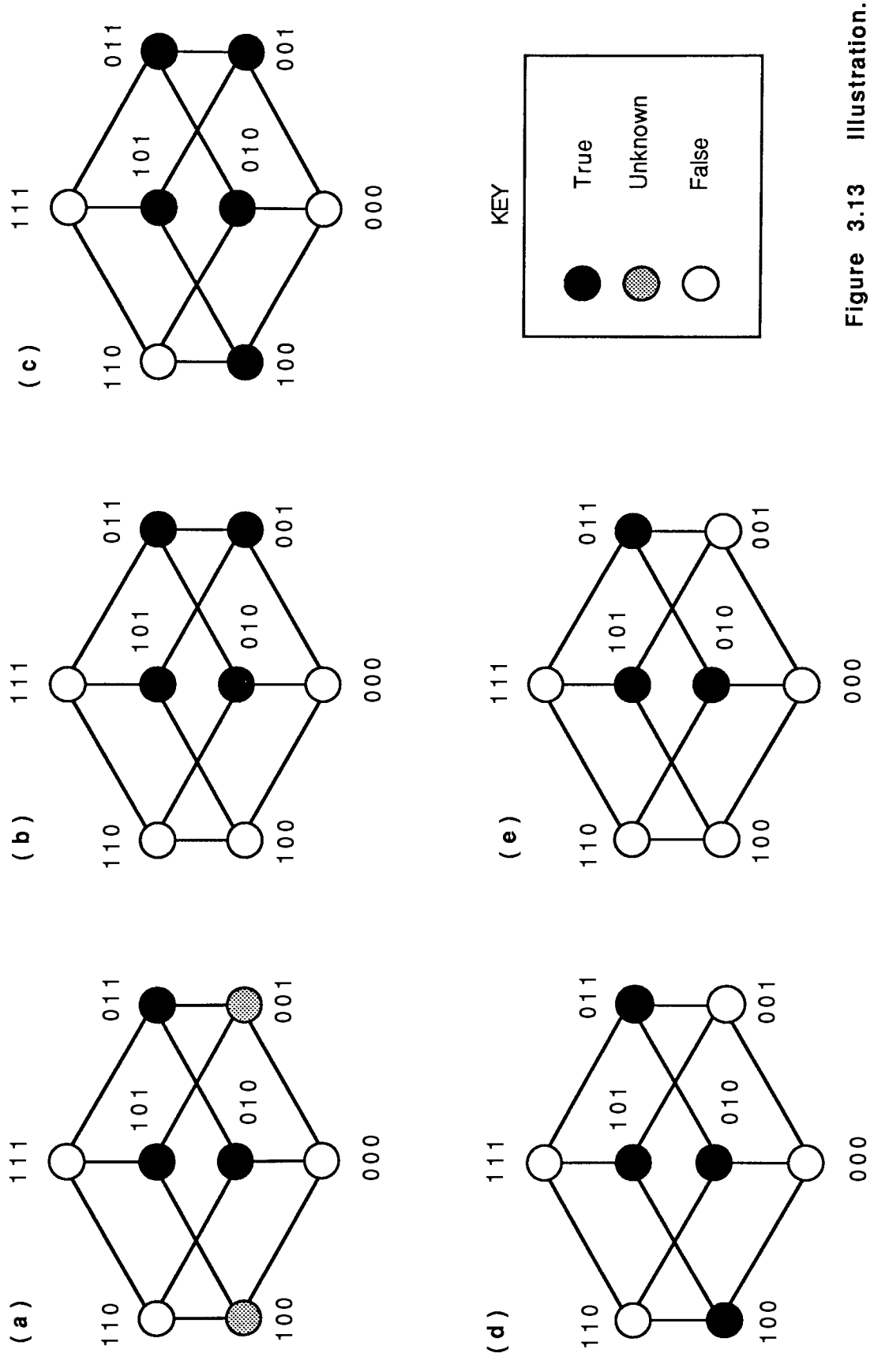


Figure 3.13 Illustration.

Is such an algorithm always decidable? What do we do if one of the neighbouring nodes is itself unknown? Rules applicable to various circumstances are found to be required in order to solve these problems. It can even be useful to consider thresholds so that the weight of evidence is substantial before making a decision. Thirdly, and worst still, the algorithm relies upon an *exponentially-expanding* checking-process *for every single node to be assessed* - and there could be millions of nodes to check!

An alternative viewpoint is given in Figure 3.13 (c) where both nodes are now constrained to be true. In this case emphasis is on positive exemplar evidence. The positive exemplars form one group minimally joined by a Mandelbrot-Peano-like multidimensional line. The positive concept, both in this case and in the general case, can be described mathematically by a minimum polynomial of inclusive disjunctives of conjunctives. Conversely, the negative concept can only be described mathematically by a minimum polynomial of mutually exclusive disjunctives of conjunctives since we have distinct groups. In this particular case, we have two such groups, the group bounded by the nodes 111 and 110 and the sole node group 000. This implies a simpler algorithm than the previous case and which lopsidedly prefers true as opposed to false nodes. That is, there is an **overgeneralisation bias towards positive evidence.**

In Figure 3.13 (d), node 001 is taken to be false and node 100 is taken to be true. The reason being that, if we consider the *whole* hypercube, this forms a symmetrical grouping of opposing lines, both positive and negative. This holistic approach is greatly recommended by the experience and history of AI where failure is often due to its absence, Boden (1977). This is the converse of case 3.13 (b) and produces a *balanced* algorithm giving equal weight to both positive and negative evidence. Since we are now considering the hypercube as a whole, rather than iteratively considering individual nodes, this gives potentially by far the fastest possible algorithm, that is, assuming that we can discover a straightforward means of holistic computation. Hence in this case we have a **symmetrical bias towards weight of *global* evidence.**

Lastly, in Figure 3.13 (e) both nodes are taken to be false. This is the negative exemplar evidence converse of Figure 3.13 (c) and hence the rationality implies an **overgeneralisation bias towards negative evidence.**

We summarise the results of the above discussion of the various biases as below.

1 Asymmetrical bias towards weight of local evidence.

2 Overgeneralisation bias towards positive evidence and overspecialisation bias towards negative evidence.

3 Symmetrical bias towards weight of global evidence.

4 Overgeneralisation bias towards negative evidence and overspecialisation bias towards positive evidence.

3.3.3 Some Psychological Experiments.

The various possibilities therefore represent the various possible biases. We already have some interesting bias cases. We can bias our generalisation towards positive evidence or towards negative evidence. Or we can consider the weight of evidence either locally or globally.

Before performing any experiment we will have some expectation of what should happen. This expectation ought to be made clear. The expectation is based upon some idea or theory. This theory must be made explicit beforehand. The experiment then tests if the theory is correct. We must also consider the design of the experiment. Fig 3.14 presents a situation in which there are 22 unknown nodes. Since the final state is boolean and

$$2^{22} = 4{,}194{,}304$$

we have a choice among approximately 4 million. This is far too many possibilities and subjects cannot visualise fifth-dimensionally anyway. Hypercubes are structurally similar in any dimension. Therefore nothing is lost in the generality of the result by lowering the dimensionality. [7] Hence, we experimented with the very much simpler third dimension and presented problems with an average of four possibilities. This keeps the problems simple. We do not want to confuse the results by overloading the subjects so that only those with good brains and considerable attention span can solve the problems.

Each problem presents a cube in which certain exemplars are known. The known exemplars are either true or false. The rest of the universe of possibilities, that is, the remaining nodes in the cube, are unknown. If there are typically four possible ways of completing the cube then these four cases represent four possible ways of generalising the known exemplars to the universe of possibilities. Logically, there is no reason to pick out any one of these four cases in the absence of external constraints. Suppose that subjects are presented with this abstract situation. Subjects are told that the diagram represents a cube, the corners of which are either black, white or grey and asked to decide whether the grey nodes should become black or white. Surely this is ridiculous? There are four possibilities and no logical reason to choose between them. How can one choose? Therefore our very reasonable theory is as follows.

Subjects will choose randomly among the four possibilities. Given a reasonable number of subjects, say six or eight, we would expect to begin to see evidence of random choice among the four possibilities. If we set a reasonable number of variations of such experiments, say six or eight, say averaging four possible solutions each, then we would

---

7 True in this case, but by no means generally true! Analogously, consider the successor generator for integers. If OLDNUMBER is initially set to zero and NEWNUMBER is given by OLDNUMBER plus one, then all positive integers are successively generated. But it is not true that all properties of number theory are apparent by examining the first application of the successor rule, namely, 0 + 1 = 1.

again expect to see evidence of random choice among the possible solutions in each of these experiments. If the subjects were in some later experiment to be told that the nodes represent something in particular then the external real world constraints would ensure that subjects would choose only one of the possibilities - maybe because the other choices would not then make any sense. In a nutshell our "**reasonable theory**" is as follows.

Because the situation is abstract and there are no external constraints, the subjects will choose randomly.

This was our confident expectation. The reader might object that these experiments are a waste of time. We are merely doing experiments to test whether, faced with a random choice, people choose randomly! Any slight variation in the result from random could be put down to the quirkiness of human nature or statistical variation. Any other result is just not going to happen. The outcome is so obvious that that the experiment is not worth while doing. On the other hand, testing for the obvious can sometimes lead to a seemingly pointless experiment providing some new insight. Testing the untested, but confidently expected obvious, can very occasionally turns up the unexpected. Like now.

The experiments demonstrated that this apparently "reasonable theory" is *wrong*. We require an **"unreasonable theory"**!

The actual experiments involved 12 subjects and each subject was set 12 problems in two sets of 6 problems. We illustrate the first set of 6 problems in Figure 3.14. The results of the psychological experiments were quite astoundingly consistent. We analyse the results by a transitivity argument. For the first set of 6 problems taken as a set there are approximately 2000 possible different solutions. Of these 2000 possibilities a particular solution set from one person has a random chance of 1 in 2000. Yet all 12 subjects gave the same solution set, see Figure 3.15 and **notice how beautiful the solution shapes are**! The possibility of this happening randomly is 1 in 2000^{12}!

The second set contained less constrained (harder) problems and again there was a very high degree of agreement (well over 99.9 %). Some of these problems had a greater number of variations than the well known psychological limit on short term memory "seven plus or minus two" (Winston, 1984). It was a mistake to have presented problems with greater than this number of variations - errors were bound to occur. Possibly also, these minor differences in the harder set could be explained away on the basis of perturbation factors such as tiredness near the end of the set, random lack of concentration, or limited visualising capabilities for these harder problems. The problems took the subjects quite some time to solve - up to half a minute in some cases. Therefore tiredness and lack of concentration with such an unusual exercise could explain slight differences near the end of the second test. Secondly the later problems required considerable three dimensional visualisation and the subjects could have lapsed into a two dimensional perspective. The

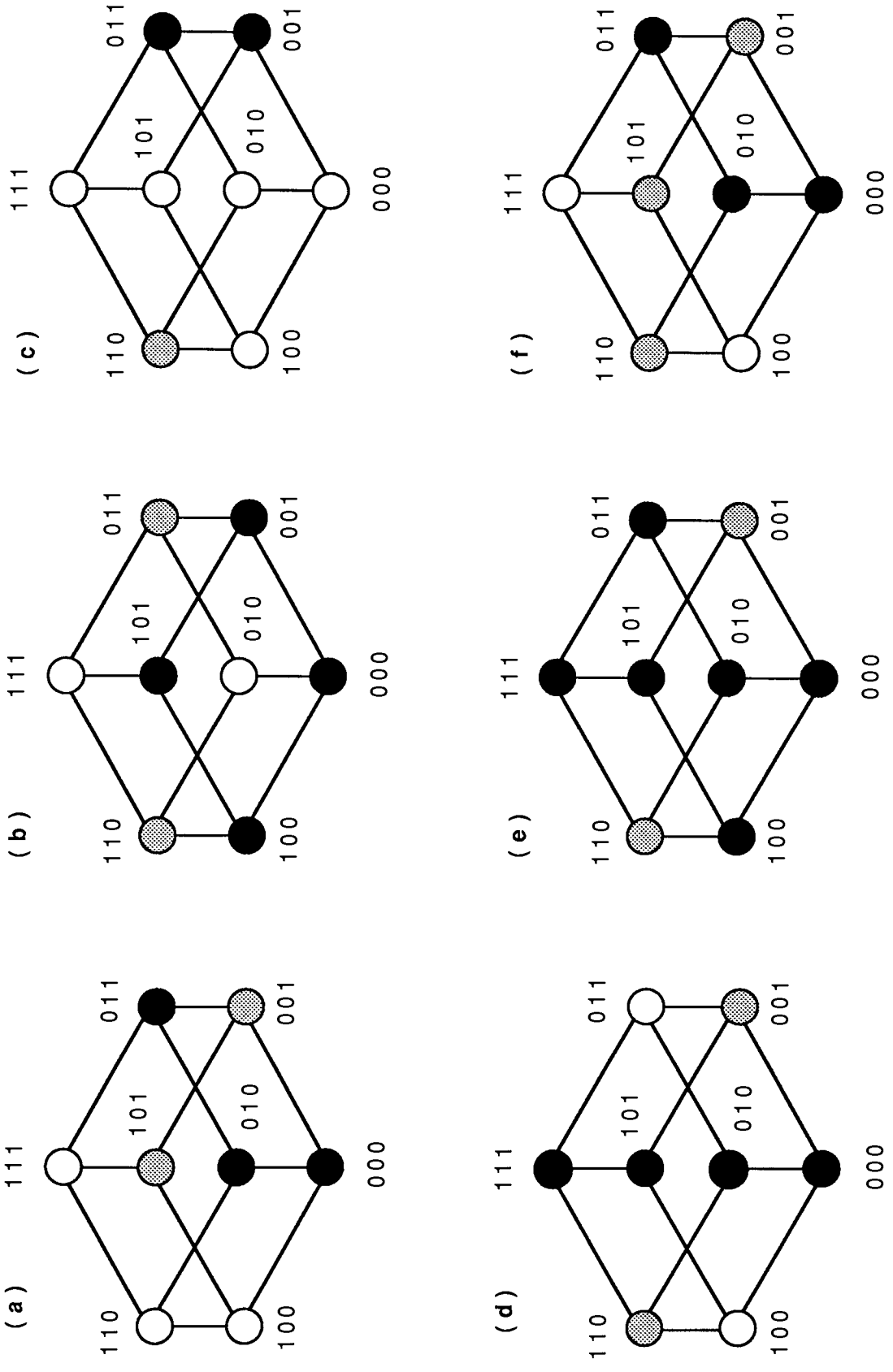


Figure 3.14 Tests 1 - 6: Questions.

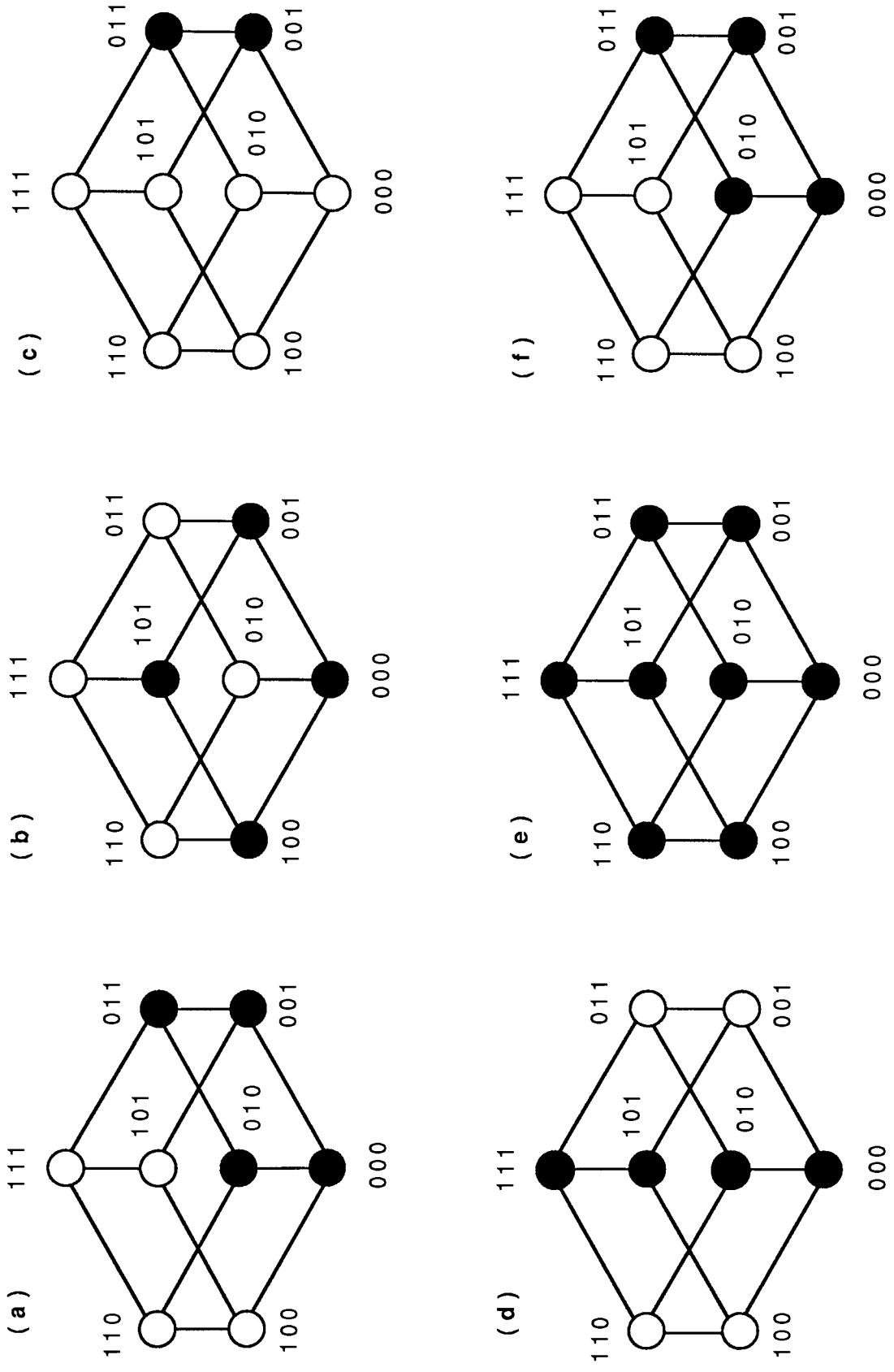


Figure 3.15 Tests 1 - 6: Replies.

subjects ranged from other researchers to motorway cafe waitresses! No detectable differences occurred with respect to the subjects "profession." The experiments could be criticised on many counts, for example:

1) There was a lack of a "control."
2) There is uncertainty in the extent to which subjects 'saw' the solution in three dimensions.
3) The failure to provide the full solution set from which a multiple choice could be made (to avoid the error of overlooking some particular choice).
4) The precise wording of the instructions, as above, given at the start could influence the result.

Nevertheless, the overwhelming consistency of the result is remarkable. Something very definite and reproducible is evidenced in these results.

Far more remarkable is the fact that the solution set that the subjects produced, when analysed for each problem, reveals itself as a consistent desire to find a global symmetry solution which **corresponds precisely with the simplest possible mathematical solution** in every case as already suggested above for Figure 3.13 (d). Since the experiments demonstrated our "reasonable theory" is incorrect we require a new theory. The new **unreasonable theory** is clear.

> *If there are no external constraints, subjects will not choose randomly among logically equal but abstract possibilities; rather, they will always very strongly prefer the simplest possible global generalisation.*

The experiments had varying numbers of exemplars. Yet each experiment led to the same simplifying preference. This suggests that there is nothing special about the number of exemplars as a proportion of the universe. The conclusion must be that if subjects prefer the simplest solution and further evidence contradicts this solution then subjects will revise their generalisation to be the next simplest possible solution in the light of the new situation which includes this new evidence. Perhaps external constraints have a similar effect. If the external constraints suggest that the simplest possible solution is impossible then the next simplest possible solution will be entertained.

We need to test this new theory. This is more complex. Merely one side issue occupies the rest of this thesis. Why? The reason is, that as stated above, our main aim is to build a first approximation to human generalisation into the hypercube machine. We therefore need to develop a theory of the hypercube and extrapolate this theory to include an approximate machine equivalent of the new "unreasonable theory." A machine needs to be built as an implementation of this extended theory. Experiments are required to test the

performance of the machine. Clearly, all this is going to occupy the rest of this project. We postulate the machine equivalent of the unreasonable theory, the **MEU theory**.

> *If there are no external constraints, the generalisation process will not choose randomly among abstract possibilities; but rather, will always choose the simplest possible globally derived generalisation.*

We must beware of getting ourselves into a circular argument. The hypercube machine is to be based on the MEU theory. If the machine gives interesting and useful performance, that does *not* prove that the unreasonable theory is correct! This is aside from other matters discussed in section 3.1, which would invalidate such an attempt anyway.

To recap, the interesting insight is the resulting mathematical description. It turns out that these descriptions correspond to the simplest possible mathematical description as we previously found by analysis in the case of Figure 3.13 (d), the global symmetry biased case.

Hence we conclude a strong preference for the simplest possible globally derived explanation that fits the data in a balanced way with respect to both positive and negative evidence. Our contender for principle 'X' is therefore nothing more than a hypercube equivalent of Occam's Razor (the principle that entities should not be multiplied beyond necessity) or as Albert Einstein was fond of saying "Everything should be as simple as possible, but not simpler" (Einstein, 1954). Perhaps we could have guessed all this without doing the psychological experiments since philosophers have been saying so much for centuries, a rare point of agreement amongst them. We did not, however, anticipate this result and looking back at the original problem that we used to start this whole discussion, namely, Figure 3.12, **it was not obvious!**

We now know the solution we require the machine to search for. In section 3.1.3 we summarised the case we had made for a sound scientific approach. The first stage in the scientific cycle for machine learning was stated as: *"We require solid links into experiments on reality concerning some fundamental, eternal fact about learning."* We have attempted to distil just such an insight into the elements of human learning and as a result we conclude that a learning machine should have a strong preference for:

> **the simplest possible globally derived explanation that fits the data in a balanced way with respect to both positive and negative evidence.**

Clearly, we now need to consider the second point in this scientific cycle. That is, what kind of an algorithm would achieve this effect? This necessitates the development of a formal hypercube theory.

3.4 HYPERCUBE THEORY.

In the set-theoretic treatment below, we take a "universal algebra" approach as, for example, is to be found in Cohn (Cohn, 1965), to a specialised lattice algebra, complemented distributed modular lattice or CDML (see Appendix 2) theory. The "father" of lattice theory was Birkhoff (Birkhoff, 1948). We create a theoretical foundation and a very wide ranging mathematical perspective on the development of the resulting Φ–boolean machine. This approach offers the possible hope of theoretically delimiting the scope of such an Artificial Intelligence system.

This "pure" and all-encompassing attempt contrasts markedly with other comparable contemporary systems where rarely has a rigorous attempt been made to define what an artificial intelligence is and is not theoretically capable of. This is, in contrast to merely defining the theoretical machine or simpler, stating the learning equation or simpler still, no theory, just "suck it and see."

The down-side of all this is twofold. Firstly, universal algebra implies working with features in common to algebras. This enables the theoretical scoping but has the disadvantage of becoming increasingly abstract and more importantly as an evolving scientific approach to mathematics is, as yet, in itself incomplete! Secondly, the latter point suggests that for any non-trivial machine the exercise may be impossible to conclude. Indeed such an attempt could alone occupy many mathematical research students. Notwithstanding, the scoping that is discovered is clearly highly desirable.

Proofs and examples are often omitted for brevity, but similar work may occasionally be discovered in the above two standard texts or in Lipschutz (Lipschutz, 1964). Our stated purpose is served in the following way: for the most part we are content with iteratively defining and then combining terms into more complex structures for later examination of their properties with respect to the hypercube. A standard notation is used throughout and listed for reference in Appendix 1. Mathematical and other definitions are also given in Appendix 2.

3.4.1 The atoms of mathematics.

Through mathematics, as in science, we can perceive **order** in the nature of things. This ordering can be used to define **structures** which enable us to understand the world. Mathematics like science has attempted to find fundamental building bricks for these structures - akin to atoms in science. The first attempt was made by Pythagoras using integers. The second attempt was by using sets. Sets are more fundamental than numbers and have proved to be a near miss. Sets are still insufficient as a basis for <u>all</u> mathematics. Hence

we immediately find a first restriction on our stated purpose with respect to theoretical scoping.

The term "set" may be used for anything that has a well-defined, that is distinguishable, collection of elements or members. A set can be defined by a membership test without reference outside itself. This has the advantage of *minimizing assumptions*. But there is a huge and subtle problem.

"Distinguishable" by who or what? Clearly, considerable intelligence and real world knowledge is required to interpret the symbols, words etc. in the requisite manner and this implies an **untold number of assumptions**! This interpretation process is trivial for man, but unwise to assume for say a seal due to the absence of a common semantic appreciation. [8] The problem is even worse in the case of a mere machine. This restricts *reliable* application of the hypercube to (human) specified and verified I/O, that is, if *we* are to appreciate the results. By this trick we avoid the flaw in set theory. In effect, the machine has a built-in **bias** or restriction - **the assumption of the applicability of set theory.** [9]

3.4.2 Sets and associativity.

We assume basic set theory as summarised below and highlight further aspects.

Idempotent	$A \cup A = A,\ A \cap A = A$
Associative	$(A \cup B) \cup X = A \cup (B \cup X),\ (A \cup B) \cup X = A \cup (B \cup X)$
Commutative	$A \cup B = B \cup A,\ A \cap B = B \cap A$
Distributive	$A \cup (B \cap X) = (A \cup B) \cap (A \cup X),$
	$A \cap (B \cup X) = (A \cap B) \cup (A \cap X)$
Identity	$A \cup \varnothing = A,\ A \cup \Theta = \Theta,\ A \cap \Theta = A,\ A \cap \varnothing = \varnothing$
Complement	$A \cup A' = \Theta,\ (A')' = A,\ A \cap A' = \varnothing,\ \Theta' = \varnothing,\ \varnothing' = \Theta$
De Morgan	$(A \cup B)' = A' \cap B',\ (A \cap B)' = A' \cup B'$

There are two ways of specifying sets. Either we may list all the members irrelevant of order or we can state some common property possessed by every element in the set and by no member (of the given universe) not in the set, for example:

{a, e, i, o, u}	is defined by listing all members (the tabular form of a set),
{x ∋ x is vowel}	is defined by a property of all members (set of all x such that = vowel)

---

8 Seals seem to have a large language yet, to date, it has proved to be uncrackable.

9 Possibly, in practice, this assumption will always hold or its exceptions be resolvable at a higher level; else we search in vain for the origins of the 'chicken and egg' riddle. Surely neurons work this way? In contrast, Einstein thought otherwise, at least, as far as concepts are concerned (Einstein, 1954).

Two sets A and B are said to be equal if they consist of the same elements. That is, every element of A is an element of B and every element of B is an element of A. Both conditions are required.

Sets can have subsets. If every element of B is an element of A then B is a subset of A or $A \supseteq B$. Certain paradoxes in set theory can be avoided by the use of hierarchies of sets eg. a family is a set of classes, a class is a set of sets and a set is a set of elements. Inheritance can be properly modelled in this manner (Touretzski, 1986). A hypercube can similarly represent a given family.

By the word **space** we mean any non-empty set which possesses some type of mathematical structure - as in vector space, metric space or topological space. Let the elements in a space be called **nodes**.

Special sets in a given space may be specified, for example, the set of all real numbers R or the set of all positive integers Z^+.

One class which commonly occurs is the set of all subsets of a given set. This is called the "power set." If $A = \{\alpha, \beta, \gamma\}$ then the power set of A is given by:

$$H^3{}_{na} = \{\varnothing, \{\alpha\}, \{\beta\}, \{\gamma\}, \{\alpha, \beta\}, \{\alpha, \gamma\}, \{\beta, \gamma\}, \{\alpha, \beta, \gamma\}\}$$

Any subset of set A is an element of the power set. Hence, if the set has n members the power set has 2^n elements since each element either does or does not belong to a given subset. This defines a hypercube. We shall call such hypercubes "**non-associative**." They have the advantage of requiring minimal memory.

In contrast "**associative**" hypercubes contain redundancy but have many advantages such as increased felicity for fast algorithms, increased orthogonality and graceful degradation. The associative hypercube equivalent of the above power set is:

$$H^3{}_a = \{\{\alpha', \beta', \gamma'\}, \{\alpha', \beta', \gamma\}, \{\alpha', \beta, \gamma'\}, \ldots, \{\alpha, \beta, \gamma'\}, \{\alpha, \beta, \gamma\}\}$$

where α', ... is the negation of α, ... respectively. This, in the general case, gives:

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{n-2} + \binom{n}{n-1} + \binom{n}{n}$$

$$\therefore \sum_{r=0}^{n} \binom{n}{r} = \sum_{r=0}^{n} \left(\frac{n!}{r!\,(n-r)!} \right) = 2^n$$

elements in the hypercube, as before. The number of flattened elements $H^n{}_{fa}$ in $H^n{}_a$ is:

$$n \sum_{r=0}^{n} \binom{n}{r}$$

The number of flattened elements H^n_{fn} in H^n_{na} is:

$$\sum_{r=1}^{n} \left(r \binom{n}{r} \right) + \binom{n}{0} * 0$$

The redundancy R is given by:

$$R = \frac{H^n_{fa}}{H^n_{fn}} = \frac{n \sum_{r=0}^{n} \binom{n}{r}}{\sum_{r=1}^{n} \left(r \binom{n}{r} \right) + \binom{n}{0} * 0}$$

$$= \frac{n \sum_{r=0}^{n} \left(\frac{n!}{r!\,(n-r)!} \right)}{\sum_{r=1}^{n} \left(\frac{n\,r\,(n-1)!}{r\,(r-1)!\,((n-1)-(r-1))!} \right)} = \frac{n2^n}{n2^{n-1}} = 2$$

Hence the advantages of the associative hypercube are bought at the price of redundancy as Table 3.1. This overhead can, in practice, be minimised by the use of virtual hypercube memory.

	n				
	1	2	3	4	...
Associative	2	8	24	64	...
Non-Associative	1	4	12	32	...

Table 3.1. Redundancy in non/associative hypercubes.

3.4.2.1 The Cartesian product.

Subsets can be represented by segments of a number line. Pairs of subsets can be represented by points in a plane. Complementarity is shown by 1) algebra where points in a plane can represent pairs of numbers and 2) geometry where pairs of numbers can represent points. Setting two number lines at right angles defines a plane corresponding to the set:

$\{(\alpha, \beta) \ni \alpha \in A, \beta \in A\}$ called the cartesian product of (set of real numbers) A with itself.

In Cartesian geometry, by convention, the horizontal and vertical axes represent the first and second elements of the ordered pair $\langle a, b \rangle$ respectively. This set is the Cartesian product of A with itself. The Cartesian product can be also found of set A with set B as in:

$$\{\langle \alpha, \beta \rangle \ni \alpha \in A, \beta \in B\}$$

We will use the symbol $\otimes$ to denote the process of taking the cross product, as in:

$$A \otimes B = \{\langle \alpha, \beta \rangle \ni \alpha \in A, \beta \in B\}$$

$A \otimes B$ is itself a set from which we could form the Cartesian product of $(A \otimes B)$ with Γ:

$$(A \otimes B) \otimes \Gamma = \{\langle\langle \alpha, \beta \rangle, \gamma \rangle \ni \alpha \in A, \beta \in B, \gamma \in \Gamma\}$$

which is again a set of ordered pairs with the first element of each pair itself an ordered pair. Conversely $\langle \alpha, \beta, \gamma \rangle$ is an ordered triple. In the general case we could have an ordered n-tuple:

$$(A \otimes B \otimes \Gamma \otimes \ldots \otimes V) = \{\langle \alpha, \beta, \gamma, \ldots, \nu \rangle \ni \alpha \in A, \beta \in B, \gamma \in \Gamma, \ldots, \nu \in V\}$$

in a multidimensional space.

3.4.3 Simple and general boolean quantities.

A simple boolean quantity can take one of only two values, 1 and 0. These symbols are however *arbitrary*. In logic the outcome of a proposition as true or false is also boolean, thus we can assign the value 1 to true and 0 to false. This is again an *arbitrary choice.*

A general boolean quantity is obtained when each component of its support set is associated with a simple boolean quantity. Conversely, a simple boolean quantity may be considered as a general boolean quantity with one element. Let upper case letters such as A, B, ... denote general boolean quantities and the corresponding subscripted lower case letters for example:

$$\{\alpha_1, \alpha_2, \ldots, \alpha_n\}, \ \{\beta_1, \beta_2, \ldots, \beta_m\}, \ \ldots$$

denote the simple boolean quantities of their corresponding components in their respective support sets. The set may be finite or infinite. We will consider only finite sets. The components may be written in the form of a matrix, either by row or column. We shall use row format as in for example:

$$B = \{\beta_1, \beta_2, \ldots, \beta_n\}$$

There are two special sets to consider when all the components have the same value either 0 or 1, denoted by O for the zero set (not to be confused with ∅, the "null set" with zero members) and I for the unit set respectively:

$$O = \{0,0, ..., 0,0\}, \qquad I = \{1,1, ..., 1,1\}.$$

3.4.3.1 Ordering.

The set of values of the simple boolean variable can be ordered by the relation $0 < 1$. It is convenient to represent the 2^n values of the general boolean variable, A, in a row format and the 0's and 1's of the associative hypercube can then be considered as binary digits. It must be remembered that this is for convenience only and is an *arbitrary choice* and implies that the possible values or "affixes" of A are 0 to 2^n -1. Equally, we could more compactly label the vertices with octal, hexadecimal or decimal equivalent integers. This could save memory space but at the expense of conversion time. The result is the general boolean function *total ordering* of affixes. This hierarchy amongst the affixes is, however, not particularly useful and as stated above arbitrary. It is found to be useful both within lattice theory and in machine learning (eg version spaces) to distinguish a lesser class, namely, partial order relations. If we extend the definition of the order relation for simple booleans then:

$$A > B \text{ if } \forall i\ (a_i >= b_i) \text{ and } A \neq B.$$

(Note: if we defined $A > B$ if $\forall i\ (a_i > b_i)$ then only O and I are involved in the relationship which is of little interest).

Further:

$$A >= B \text{ if } \forall i\ (a_i >= b_i).$$

Hence:

$$O <= A <= I \text{ and } O <= B <= I.$$

The arcs of the cube in Figure 3.2 (c) are readily seen to define the partial order by "monotonic levels." Given nodes A and B, if it is possible to go from A to B by monotonically either moving up or down the hypercube, then $A > B$ or $B > A$ accordingly. Otherwise no relationship is defined, as in 1,1,0 and 1,0,1.

A boolean quantity $\psi 1$ (either simple, α or general, A) is a function (either simple, f or general, F) of another boolean quantity $\psi 2$ (either simple, β or general, B) if there is a well defined value of $\psi 1$ for each and every value of $\psi 2$. Hence we obtain 4 cases:

$$\beta = f(\alpha), \qquad \beta = f(A), \qquad B = f(\alpha), \qquad B = F(A).$$

For example, we might have f(A) = 1. That is, we understand the value of the function at all other nodes in $\Theta = 0$.

3.4.3.2 ϕ-boolean quantities.

Let us now consider the general Φ-boolean quantity which has at least some components represented by the "don't care" or unknown state, ϕ. The variable takes one of three values 0, ϕ, 1. Consider $f(\Phi) = 1$. We have here an analogy with the corresponding neural network where Φ represents the input, f represents the hidden layers and 1 is a single node boolean output. A typical convention in neural nets is that an output value of '1' identifies the function.

Similarly, the **weight** of Φ as given by the number of nodes in its representative set has considerable analogy with the term "weight" in neural networks. If the weight is normalised in both nets for the same problem then we postulate that the analogy becomes exact in the particular case of the global minimal potential energy for both the neural net and the hypercube - since in the hypercube we are always ultimately considering the nature of Φ.

3.4.3.3 Duality and the complement.

The **dual** α^* of the simple boolean variable α is obtained by mapping the set of values of the variable onto itself such that 0 -> 1, 1 -> 0. Hence symmetry is preserved as in:

$$(\alpha^*)^* = \alpha.$$

If $\alpha_1 > \alpha_2$ then $(\alpha_1)^* < (\alpha_2)^*$.
Similarly for a general boolean A, $\alpha_i \rightarrow \alpha_i^*$.
Hence:

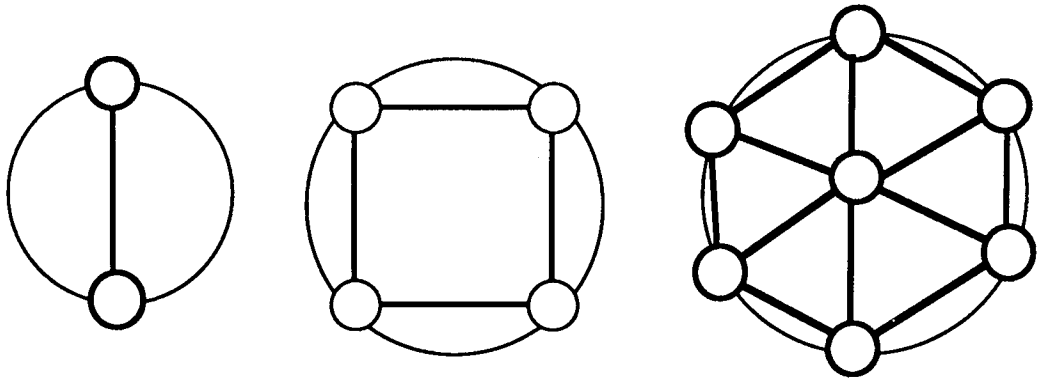
$$(A^*)^* = A,\ I^* = O,\ O^* = I,\ A >= B \rightarrow A^* <= B^*.$$

This preservation of symmetry has an important interpretation. Duality can be seen as symmetry about the centre of the hypercube. Three points are in order.

1) The centre of the hypercube is undefined within the hypercube formalism. We must consider its virtual equivalent. There is a second virtual centre at infinity.

2) Careful consideration of the above implies that the hypercube is not just missing its centre but in fact has no inside at all. Hence $\forall$ n in H^n all nodes lie on the outside. It follows that the thickness of the "skin" of the hypercube is dimensionally infinitesimal.

3) These two points suggest an alternative interpretation of the hypercube, namely, that all nodes lie on a hypersphere S^n whose major centre is the virtual centre of the equivalent hypercube, as Figure 3.16. This can, at times, be a useful way of looking at problems.



(a) (b) (c)

Figure 3.16. The S^1 to S^3 hyperspheres.

An example of a Φ-boolean hypersphere in S^5 is given in Figure 3.17.

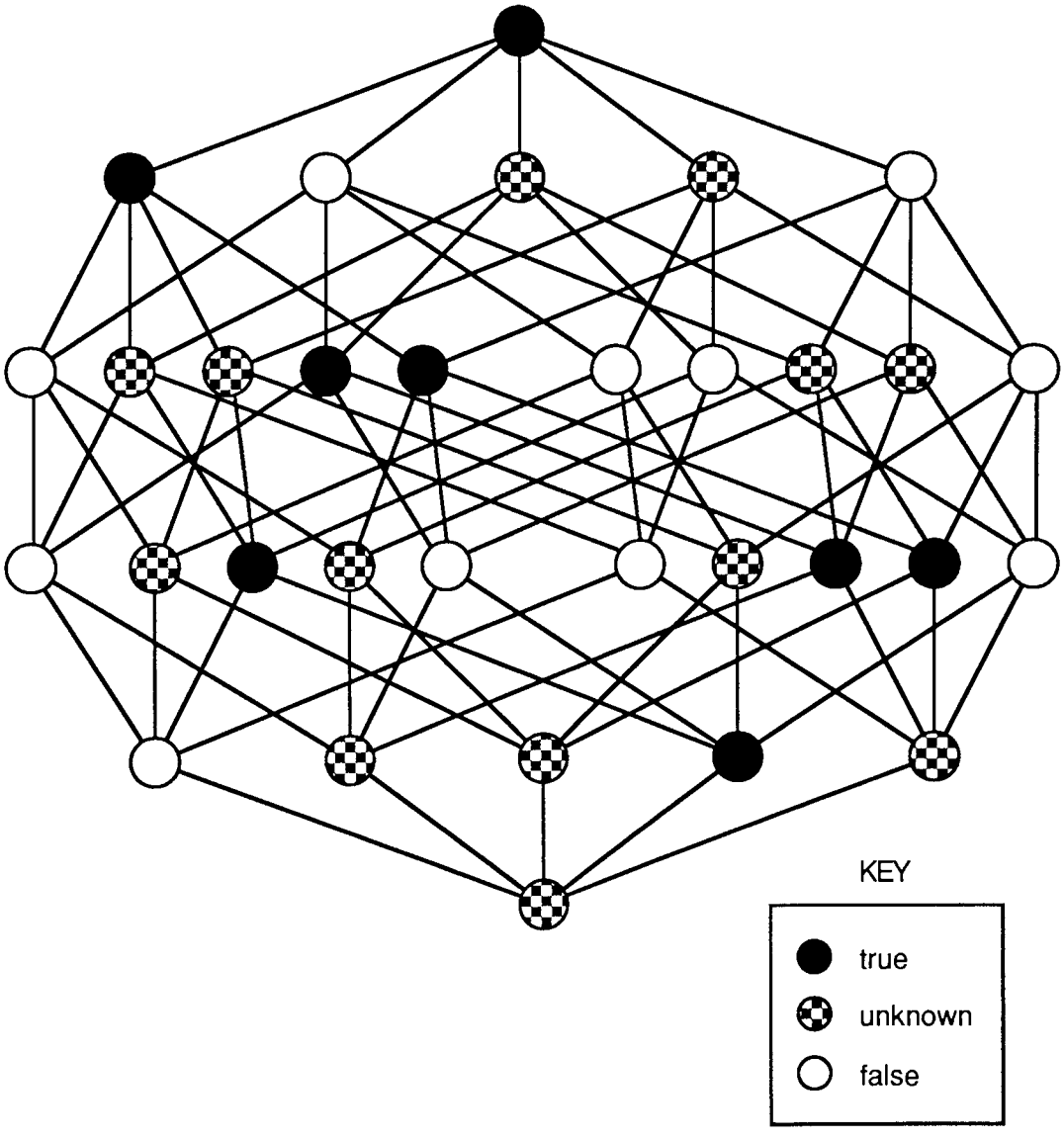


Figure 3.17. A Φ-boolean hypersphere in S^5.

We can also consider the duality $F^*(A)$ of a boolean function F(A) where:

$$F^*(A) = (F(A^*))^*.$$

Again, the following identity holds:

$$F^{**}(A) = F(A),$$

since: $((F((A^*)^*))^*)^* = F(A)$.

Consider the number of general boolean functions of n variables. If B = F(A) where A has n components and B has m components then A has 2^n states for each of which B can take 2^m states. Hence there are:

$$(2^n)$$
$$(2^m)$$

different possible boolean functions. We tabulate m against n for low values in Table 3.2. It is convenient to use the terms **index** (to represent the connectionism of a particular node) and **class** (for the classification of the result of the function) for n and m respectively.

		n			
		1	2	3	4
m	1	4	16	256	65536
	2	16	256	65536	...
	3	64	4096	...	...
	4	256	65536	...	...

Table 3.2. Binary powers table of mn states.

Due to the explosive nature of larger m values we restrict our consideration to hypercubes where m = 1. This need not necessarily be a restriction in practice. For example, from Table 3.2: (m, n) = (4, 1) = 256 = (1, 3) = (m, n). Considering the first cell (1, 1) in Table 3.2, there are four simple functions of one simple variable as in Table 3.3.

α	f1	f2	f3	f4
0	0	0	1	1
1	0	1	0	1

Table 3.3. The functions of a single variable.

Note that $f_1 = O$, $f_2 = \alpha$, $f_4 = I$, and $f_3 = \alpha'$.

That is, f_3 has the special property that it is always the opposite or **complement** α' of the variable α. Clearly:

$(\alpha')' = \alpha$, $\alpha' = \alpha^*$ and $\alpha > \beta$ in the complementary form is $\alpha' < \beta'$

Similarly, we can consider the complement A' of the general boolean variable A with components α_i' and α_i respectively. Hence $A^* = A'$, $(A')' = A$. Further, we can define the complementary boolean function: $F'(A) = (F(A))'$ (Note that $F'(A)$ does not equal $F^*(A)$). We can derive a number of further identities. Given two simple boolean variables α, β their boolean sum $\alpha + \beta$ is the simple union or OR function $a \cup b$. It follows that:

1) $\alpha + \beta = \max(\alpha, \beta)$. From this:
$\alpha + 0 = a$, $\alpha + 1 = 1$, $\alpha + \alpha = \alpha$, $\alpha + \alpha^* = \alpha + \alpha' = 1$.

2) $\alpha + \beta = \beta + \alpha$ (commutativity)

3) $\alpha > \beta \Rightarrow \alpha \cup \alpha >= \beta \cup \alpha$ (the equality is required)

The sum can be generalised pairwise associatively. Disregarding brackets the sum of n terms:

$$\bigcup_{i=1}^{n} (\alpha_i)$$

By the above commutativity the ordering is arbitrary hence zero terms and repeating terms may be omitted thereby simplifying the hypercube. Similarly a term equal to 1 or a complement of any other term implies that the sum is 1, again simplifying the hypercube. In general:

$$\sum_{i=1}^{n} (\alpha_i) = \max_{i=1}^{n} (\alpha_i)$$

The product properties can all be derived from the sum. We define the product $\alpha\beta$ of two simple boolean quantities α and b by the simple intersection or AND function $a \cap b$. Hence analogously to the sum:

1) $\alpha\beta = \min(\alpha,\beta)$ hence $\alpha 0 = 0$, $\alpha 1 = \alpha$, $\alpha\alpha = \alpha$, $\alpha\alpha^* = \alpha\alpha' = 0$

2) $\alpha\beta = \beta\alpha$ (commutativity)

3) $\alpha < \beta \Rightarrow \alpha\alpha <= \alpha\beta$ (equality again required).

The sum and product can be combined in the identity:

$\alpha(\beta + \zeta) = \alpha\beta + \alpha\zeta$ (distributivity).

Hence **products of sums** can be expanded. Conversely to add a product to a term, add each factor to the term and form the product of the sums. Again we may consider simplifying repeating terms, terms containing complements. Generalising the product by associativity of multiplication and dropping the brackets, the product of n terms is defined:

$$\prod_{i=1}^{n} (\alpha_i)$$

since the grouping order is disregardable and arbitrary due to commutativity. In the general case:

$$\prod_{i=1}^{n} (\alpha_i) = \max_{i=1}^{n} (\alpha_i)$$

Following on from these studies we proceeded to develop our theory of hypercube learning as discussed in the remainder of this chapter. This theory and the ensuing practical work as discussed in chapter 4 is novel except where otherwise indicated.

3.4.4 The complexity range to parity.

Consider the simple boolean functions of two simple variables as in Table 3.4.

αβ	00	01	10	11	Function	Grouping
f0	0	0	0	0	$\alpha\alpha' = 0$	a1
f1	0	0	0	1	$\alpha\beta = \eta'$	b4
f2	0	0	1	0	$\alpha\beta' = \mu'$	c4
f3	0	0	1	1	$\alpha = \alpha$	a3
f4	0	1	0	0	$\alpha'\beta = \lambda'$	c2
f5	0	1	0	1	$\beta = \beta$	a5
f6	0	1	1	0	$\alpha'\beta + \alpha\beta' = \rho$	b5
f7	0	1	1	1	$\alpha + \beta = \delta$	b1
f8	1	0	0	0	$\alpha'\beta' = \delta'$	b2
f9	1	0	0	1	$\alpha\beta + \alpha'\beta' = \rho'$	b6
f10	1	0	1	0	$\beta' = \beta'$	a6
f11	1	0	1	1	$\alpha + \beta' = \lambda$	c1
f12	1	1	0	0	$\alpha' = \alpha'$	a4
f13	1	1	0	1	$\alpha' + \beta = \mu$	c3
f14	1	1	1	0	$\alpha' + \beta' = \eta$	b3
f15	1	1	1	1	$\alpha + \alpha' = 1$	a2

Table 3.4. The functions of two variables.

The example is illustrative of a number of important points. Many subgroups can be found particularly for higher H^n. Some of these subgroups are of particular interest. In group 'b' the 6 symmetric functions break down into two subgroups: b_5 and b_6 are two parity operations involving the full sum and product, and b_1 to b_3 inclusive are four non-parity symmetry operations involving either the sum or the product. There are always two parity cases in any H^n and they correspond to: even parity or the exclusive OR or the non-equivalence operation, and odd parity or the exclusive NOR or the equivalence operation. In further detail:

1) All of the functions in the function column of Table 3.4 are expressible in terms of the sum, product and complement. Two functions result in constants.

2) There are three groups:

a) 6 functions depending upon only *one variable:* (a_1 - a_6)

$\alpha,\ \alpha',\ \alpha\alpha',\ \alpha + \alpha',\ \beta,\ \beta'$

designated as designator:

$\alpha,\ \alpha',\ 0,\ 1,\ \beta,\ \beta'$

b) 6 *symmetric* functions designator: (b_1 - b_6)

$\alpha + \beta,\ \alpha'\beta',\ \alpha\beta,\ \alpha' + \beta',$

$\alpha'\beta + \alpha\beta',\ \alpha\beta + \alpha'\beta'$

designated as designator:

$\delta,\ \delta',\ \eta',\ \eta,\ \rho,\ \rho'$

c) 4 *non-symmetric* functions: (c_1 - c_4)

$\alpha + \beta',\ \alpha'\beta,\ \alpha' + \beta,\ \alpha\beta'$

designated as designator:

$\lambda,\ \lambda',\ \mu,\ \mu'$

It is useful to extract a **parity distance table** from the above as Table 3.5. Table 3.5 is generated as follows. Being the functions of two variables the first five columns are the same as in Table 3.4. The last column is the **grouping** as in Table 3.4 and headed "gr." The second from last column is the associated **designator** (as in point 2 above) from the right hand side of the identity in column 6 of Table 3.4 and labelled at the head of the column by "des." *(In order to appreciate the following theory the reader should study Table 3.4 with regard to the careful design and irrefutability in the logic of the designator identities).*

The sixth column is the **parity minimum distance** from "ρ" and labelled "pmρ." Locating b_5 in the grouping column "gr" we see that the designator alongside and preceding it is ρ. Looking along that column we note the bit pattern in columns 2 to 5 inclusive is 0110. The column pmρ computes the **Hamming distance**, given by the function H, from any given function to the parity pattern ρ. This distance measure corresponds in binary terms to

the number of bits that need to be flipped in order to change the function from either of the two parity functions to the given function. Therefore there are two parity distance measures.

For some node R: pmρ = H (| R – ρ |). Thus, in the first case, for function f_0, the minimum number of bits to be altered from 0000 to 0110 is 2. Hence pmρ = 2 in this instance. Similarly pmρ' is the metric for the Hamming distance from ρ'.

αβ	00	01	10	11	pmρ	pmρ'	pmin	bc	cr	subh	des	gr
f0	0	0	0	0	2	2	2	0	2·5	2·5	0	a1
f1	0	0	0	1	3	1	1	1	1	1	η'	b4
f2	0	0	1	0	1	3	1	1	1	1	μ'	c4
f3	0	0	1	1	2	2	2	2	2	2	α	a3
f4	0	1	0	0	1	3	1	1	1	1	λ'	c2
f5	0	1	0	1	2	2	2	2	2	2	β	a5
f6	0	1	1	0	0	4	0	2	0·5	0·5	ρ	b5
f7	0	1	1	1	1	3	1	3	1·5	1·5	δ	b1
f8	1	0	0	0	3	1	1	1	1	1	δ	b2
f9	1	0	0	1	4	0	0	2	0·5	0·5	ρ'	b6
f10	1	0	1	0	2	2	2	2	2	2	β'	a6
f11	1	0	1	1	3	1	1	3	1·5	1·5	λ	c1
f12	1	1	0	0	2	2	2	2	2	2	α'	a4
f13	1	1	0	1	3	1	1	3	1·5	1·5	μ	c3
f14	1	1	1	0	1	3	1	3	1·5	1·5	η	b3
f15	1	1	1	1	2	2	2	4	2·5	2·5	1	a2

Table 3.5. Parity distance complexity range.

The range ρ to ρ' gives the range of all possible boolean functions within any given hypercube. As we consider nodes which are further in Hamming distance from ρ then they will be closer to ρ' and vice-versa. In Figure 3.18, R and R' have the same absolute distance from the ρ, ρ' parity datum as given by: pmin = min (pmρ, pmρ').

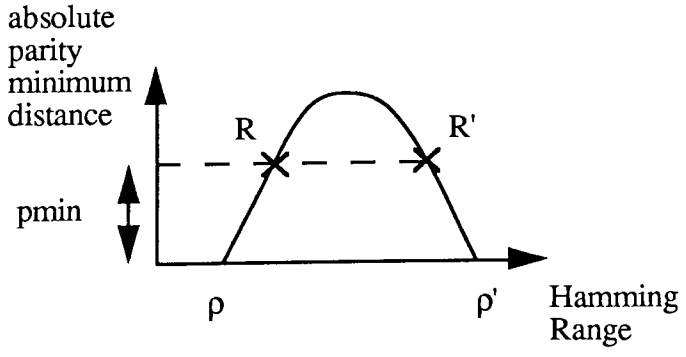


Figure 3.18. The duality of pmin for the Hamming range.

The eighth column labelled pmin is the minimum Hamming distance from ρ and ρ'. Hence by recording the least value of pmρ and pmρ' we have a measure of how close the function is to the closest odd or even parity. Since parity represents the most complex of all possible functions in any given universe represented by the hypercube, it follows that for any given function **pmin is a measure of the complexity of the function.** We see that pmin is divided into three groups: 0, 1 and 2 by value. This defines three **ordered levels of complexity** for a function in two variables.

We see, for example, that level 2 in pmin defines *all* the "a" groupings. This is, however, unsatisfactory since for example, a_2 and a_3 have the same level of complexity even though a_2 defines the whole universe as uniformly consistent and all 1's, while a_3 defines the first level subhypercube, *only*, with these properties. Common sense suggests that the two functions are not of the same level of complexity. We therefore take pmin as only a rather crude ordering of complexity. We require a refinement to further order the subtypes.

Column nine computes the raw bit count, labelled "bc", among the functions assuming positive logic. We then compute column ten the **complexity range** labelled "cr" in Table 3.5 to further separate out the intermediate subtypes of functions:

$$cr = \frac{| bc - pmin |}{bc_{max}} + pmin$$

The complexity range now correctly **orders** function subtypes as well and consistently *delineates and groups the complexity range of functions*. The complexity range in this instance works out to be five from the cr values: 0.5, 1.0, 1.5, 2.0 and 2.5. We see the necessary uniformity in the result.

The cr = 0.5 are the *only* two parity functions. *All* the cr = 1 are complemented: η', μ', λ', δ'. *All* cr = 1.5 are uncomplemented: η, μ, λ, δ. *All* cr = 2 are all the single true variables both complemented and uncomplemented. The cr = 2.5 are the *only* two constants. Comfortingly uniform!

3.4.4.1 Complexity analysis of the hypercube.

Our "complexity range" appears to correspond to the somewhat obscure topological measure of complexity referred to in Minsky and Papert (1988). Given H^n and a function within H^n then merely by computing "cr" we have a measure of the difficulty of finding the function within the hypercube. A useful exercise would be to test this theory in practice in the algorithmic implementation of the theory. In the above example in Table 3.5, functions with pmin = 2 should compute faster than those of pmin = 1 and these in turn should compute faster than the two parity functions themselves where pmin = 0. Within a given group the

computation times although similar would be expected to vary somewhat by subgroup, depending upon practical matters such as the starting point of the algorithm, etc. Further, since learning such functions is clearly exponential in nature in the hypercube, we would expect practical verification of a such curve.

We can now work through the 16 functions of $\alpha\beta$ and add up the number of functions within each type as Γ. Figure 3.19 symmetrically plots Γ against cr. The horizontal axis gives the 5 **ordered levels** within the complexity range from parity (ρ, ρ') at cr = 0.5 to the complete hypercube at cr = 2.5. Isolated nodes are functions with cr = 1.0, pairs of lines at cr = 1.5, and lines at cr = 2.

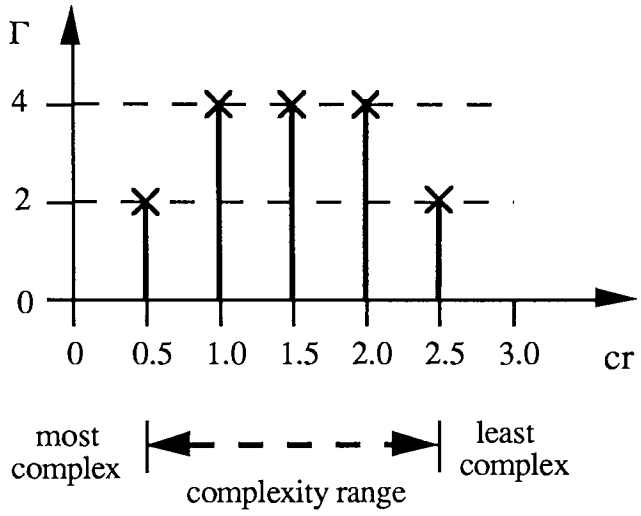


Figure 3.19. Function type number within the complexity range.

An interesting question is what is the probability of any given function taking a long time to compute? Consider parity itself. As H^n tends towards infinity the likelihood of a random parity function rapidly diminishes. For H^{10} there are approximately 10^{300} functions, yet the extremities of the complexity range have by then become insignificant. For example, there are still only 2 parity functions so the possibility of random parity is only approximately 10^{-300}. So for all practical problems such computations may never occur.

It is more useful to consider the typical problem. How hard is it to find the typical function and with what sort of spread? The vast majority of functions lie at intermediate complexity, as is true even at H^2 in Figure 3.19. The mean and mode are both at the half range due to the symmetry, in this case where cr = 1.5. Hence for a random function its most likely complexity is in the intermediate region, and for practical problems of large H^n, close to the mean.

In more detail, normalising the complexity range in Figure 3.19 for various dimensions reveals that the distribution rapidly goes from platykurtic (box shaped), through mesokurtic (normal distribution shaped) to leptokurtic (sharp peaked). It follows that for

higher dimensional problems the solution is most likely to be of intermediate complexity. The complexity range can alternatively be regarded as the generalisation (or linearity) to specialisation (or non-linearity) range. The complete hypercube (as all ones or all zeros) represents the ultimate overgeneralisation. Parity represents the ultimate in overspecialisation.

It is interesting to compare this theory with neural nets. If a neural net has a large number of neurons in the hidden layer then it tends to generalise poorly. Overspecialisation can be seen in that nodes in the hidden layer will tend to represent each of the exemplars rather than some aspect of their commonality.

Analogously, although many trees may have been seen a new example of a tree may not be recognised as such if it varies only slightly from previous examples. This is tending towards rote learning (overdependence on the teacher). Overgeneralisation is the reverse case (too independent of the teacher). Given a few examples of trees it may then recognise as a tree things which are not trees. What do we really want? Do we require a rough answer with minimal resources or a more nearly exact answer with maximal resources? In practice the answer may depend on the situation, but in general the answer is clear. This work on complexity suggests that an intermediate solution will apply in the typical case. That is, the solution will tend very often to be half way down through the subhypercubes as the dimensionality increases. Hence some **polytope** or grouping of intermediate level subhypercubes will most often correctly represent the global minimum of the universe.

An interesting research project would be to discover if there is any complexity consistency in real world problems of various types. For example, we strongly suspect (from practical work with the hypercube) that natural phenomena - as in scientific laws, mathematics, etc. - is close to parity; whereas human generated information is close to the least complex end of the range. For example, we have discovered that "history taking" in general medical practice seems to typically involve functions close to H^n such as the majority function. (Time has not allowed us to seriously delve further into this aspect).

Having some measure of the complexity of the problems strongly suggests by algorithmic analysis a relative measure of how long an algorithmic implementation of the theory will take to run and what its memory requirements will be. This contrasts sharply with neural nets such as bp where there is no such revelation from the theory.

We have considered a particular hypercube in detail, yet the **hypercube is totally orthogonal.** What is true for one dimensionality is true of the next. [10] The above theory is independent of the particular hypercube H^n and we have worked out the general case for the

10 This is not strictly speaking true since we have found minor differences in particular circumstances. Nevertheless none of these special cases affect this argument.

solution for *any* dimensionality. In the general case the **number of new functions,** Γ, depending on n variables of class c, is given by:

$$\Gamma_n^c = \sum_{r=0}^{n} \left((-1)^1 \binom{r}{n} 2^{c2^n} \right)$$

$$= (-1)^0 \binom{0}{n} 2^{c2^n} + (-1)^1 \binom{1}{n} 2^{c2^{(n-1)}} + (-1)^2 \binom{2}{n} 2^{c2^{(n-2)}} + \ldots$$

$$\ldots + (-1)^{(n-1)} \binom{(n-1)}{n} 2^{c2^{(n-(n-1))}} + (-1)^n \binom{n}{n} 2^{c2^0}$$

In the case of c = 1 we have:

$$\Gamma_n^1 = \sum_{r=0}^{n} \left((-1)^1 \binom{r}{n} 2^{2^n} \right)$$

$$= (-1)^0 \binom{0}{n} 2^{2^n} + (-1)^1 \binom{1}{n} 2^{2^{(n-1)}} + (-1)^2 \binom{2}{n} 2^{2^{(n-2)}} + \ldots$$

$$\ldots + (-1)^{(n-1)} \binom{(n-1)}{n} 2^{2^{(n-(n-1))}} + (-1)^n \binom{n}{n} 2^{2^0}$$

For n = 0 to 4 this results in 2, 2, 10, 218, 64594 functions respectively. See Table 3.6 for more detail of these first few functions. Note that * in the designation column implies the dual of the given "type."

0 variables		Total = 2 functions
Designation	Type	Number of function types
0A1	0	1
0A1*	1	1

1 variables		Total = 2 functions
Designation	Type	Number of function types
1A1	a	2

2 variables		Total = 10 functions
Designation	Type	Number of function types
2A1	$\alpha\beta$	4
2A1*	$\alpha\vee\beta$	4
2A2	$\alpha\beta\vee\alpha'\beta'$	2

3 variables		Total = 218 functions
Designation	Type	Number of function types
3A1	αβγ	8
3A1*	αβγ	8
3A2	δ(αβνα′β′)	12
3A2*	δ(αβνα′β′)	12
3A3	αβδ′να′β′δ′	4
3A3*	αβδ′να′β′δ′	4
3A4	δ(ανβ)	24
3A4*	δ(ανβ)	24
3A5	αβ′δνβδ′	24
3A5*	αβ′δνβδ′	24
3A6	αβ′δ′να′βδ′να′β′δ	8
3A6*	αβ′δ′να′βδ′να′β′δ	8
3A7	αβνβδνδα	8
3A8	αβδ′ναβ′δνα′βδνα′β′δ′	2
3A9	αβνβδνδα′	24
3A10	αβδνα′δ′νβ′δ′	24

Table 3.6. List of function types for 0 to 3 variables.

3.4.4.2 Advantages and disadvantages of cr.

The above discussion shows that the huge **advantage gained** (from the complexity analysis, the uniform structure of the representation and its boolean memory) is that ***the hypercube as a learning engine is mathematically analysable in depth.*** We can understand what is going on. Learning is no longer a black art or mystery where it is unknown what it is doing, whether it has "got stuck", how long the process will take or how good the answer will be anyway after a certain processing time. Many other advantages follow, such as an ability to cope with the tricky problems of invariance and variations on a theme. Regarding its "brute force approach", firstly it must, in theory at least, always work and secondly we can point to neurological evidence that our neurons also appear to use "brute" force rather than clever tricks. Indeed, this very insight towards "brute force" and away from the clever tricks of AI is behind the present connectionist trend.

The **difficulty** is how to compute cr in the general case, since **the general problem is NP.** (See Appendix 3 for a discussion of P and NP). We have, in fact, taken "brute force" to the extreme to get the maximum advantage of transparency. Direct application of the equation for cr will not lead to a practical algorithm, since it has to:

a) deal with *all* of the individual nodes of a particular function type in the complete universe in order to compare each node in turn against both complete parity functions, which is *very* expensive and allows the solution of only trivial problems,

b) implies storage of possibly *billions* of connections, and

c) therefore does not refer directly to the *target* global function(s) of the given problem.

3.4.4.3 Algorithmic requirements.

We require an algorithm whose results are in 1 : 1 correspondence with the above theory, yet does not have these deficiencies. That is, we have the following **requirements:-**

a) We require a strong preference for the **simplest** possible **globally** derived explanation that fits the data in a **balanced** way with respect to both positive and negative evidence (section 3.3.3).

b) We require a 1 : 1 **correspondence** with cr for all possible functions in the universe Θ, so that, by correspondence with cr, the algorithm is mathematically analysable in depth, yet *does* refer directly to the **target** global function(s) of the given problem (section 3.4.4.2).

c) We require a *very* much **faster** algorithm to fight the NP problem (for as long as possible) by considering *groups* of nodes (comprising perhaps millions of nodes at a time) *in parallel* (see Appendix 4 for a discussion of the possibility of hypercube parallelism), rather than simply considering individual nodes one at a time as in cr (section 3.4.4.2).

d) We require **virtual** storage of the non-exemplars as a connectionist sparsity within the hypercube in order to avoid memory overflow in the general case (section 3.4.4.2).

e) We require **actual** storage of exemplars and their associated classification in order that the algorithm may learn (generalise) from these exemplars (section 3.4.4.1).

f) We require, (consequently from point c in section 3.4.4.2), the use of **ϕ-booleans** in order that the algorithm should cope with groups of nodes (generalisation) as well as individual nodes (specialisation).

g) We require an **associative** hypercube for its advantages (section 3.4.2).

h) We need some flexible means of adjusting the depth of search or **"resolution"** according to circumstances. (This is analogous to flexibly altering the number of nodes in the hidden layer of a neural net until a sufficient solution is achieved). The advantage of

resolution is that it will further speed up very considerably the algorithm and make accessible the solution of very large problems (section 3.4.4.2).

We summarise these ***requirements as a quick reference single word list,*** as follows:

1) simplest
2) global
3) balanced
4) correspondence
5) targeted
6) faster
7) virtual
8) actual
9) ϕ-boolean
10) associative
11) resolution.

3.4.5 Subhypercube minimum polynomials.

Consider the geometrical representation of functions in a lattice of n-dimensional hyperspace. In this representation, as stated, a line can be considered as a one-dimensional cube, a square as a two-dimensional cube, a cube as a three-dimensional cube and cubes of four or more dimensions are called hypercubes, hence we have H^1, H^2, . . . , H^n. *Minimum polynomials* are represented by points or nodes in n-dimensional space connected by arcs (lines) defining hypercubes.

For example, if there is only a single variable, x say, then the minimum polynomials correspond to the two-element states x' and x having the binary representations 0 and 1. The line joining the two nodes is a one-cube representing the join of the two elements. In H^2 with variables x and y there are four minimum polynomials, viz:

$$x \cap y, \quad x' \cap y, \quad x \cap y', \quad x' \cap y'.$$

The four nodes corresponding to the minimum polynomials define a square as Figure 3.20.

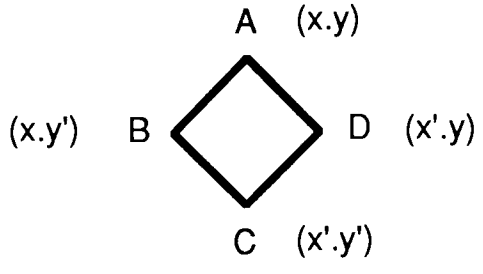


Figure 3.20 Minimum polynomials for H^2.

A side of the square is a *face* of the hypercube representing the join of minimum polynomials of the two equivalent nodes, such that side AB represents:

$$(x \cap y') \cup (x \cap y) = x.$$

Diagonals are represented by irreducible joins of minimum polynomials eg diagonal AC is represented by:

$$(x \cap y) \cup (x' \cap y').$$

Thus nodes AB form a group connected by a face, whereas nodes AC form two separate groups. The face AB represents a first level *subhypercube*. Several principles and concepts can now be defined, some examples of which are briefly discussed below. The *complement* of any function is given by the join of those nodes not incorporated in the geometrical representation of the function. For example, for a function in three variables, if we have:

$$\emptyset(x, y, z) = (x' \cap y') \cup (y \cap z') \cup (x \cap z),$$

then the complement is the remaining two nodes of the cube, viz:

$$(x \cup y) \cap (y' \cup z) \cap (x' \cup z').$$

Sometimes it is more economical to use the complement form. In any case minimisation must proceed by careful and appropriate selection of faces, since for example a pair of face diagonals will not lead to optimum minimisation. This problem is avoided by the use of subhypercubes. Further, by an arrangement of Pascal's triangle it is possible to immediately obtain the number of nodes, edges, faces, cubes and hypercubes for a given number of elements.

An important property of these n-space functions is that adjacent nodes differ in respect of only one element state, since this permits the minimisation process to be carried out via a process of subhypercubing. We may thus define a *distance* concept as the number of element-states which must be changed in passing from one node to another. This distance is equal to the minimum number of edges to be traversed in passing between two nodes. A Euclidean metric, which is rather expensive, is:

$$d(\Delta_1, \Delta_2) = \sqrt{\{\Sigma_n(\delta_{1,i} - \delta_{1,i})\}}$$

However a cruder but sufficient and cheaper metric is:

$$d(\Delta_1, \Delta_2) = \Sigma_n (|\delta_{1,i} - \delta_{1,i}|)$$

A symmetry concept is defined as any 1:1 transformation of the cube which leaves distances between all pairs of nodes unaltered. There are 2^n n! symmetries in the hypercube. For example, for H^2 there are 8 symmetries, namely, 3 rotative involving $\pi/2$, π, and $3\pi/2$ radians and 4 reflective symmetries in the horizontal and vertical axis and an identity symmetry leaving each node unchanged. These properties can be used to advantage in reducing complexity. Subhypercubes are found to form *spheres* of some radius as subsets of the lattice.

The face AB in Figure 3.20. represents a first level subhypercube. It follows from basic geometrical considerations that an equivalent to selecting nodes as representing minimum polynomials is to select *faces* to represent element-states.

These principles and a number of other concepts can be extended to higher dimensions and the problem of finding the logical representation of a group or groups within the lattice reduces to a **search through the subhypercubes.**

3.4.5.1 Learning from examples as a search.

Inductive learning from examples can be viewed as carrying out a *search* through a space of all possible concepts (Mitchell, 1982). This "hypothesis space" exists in a number of dimensions, hence the validity of a hypercube lattice as a knowledge acquisition tool. Consider the problem of learning to distinguish between diseased cells and healthy ones. Consider a notation where each cell contains two bodies and each of these bodies has three attributes. Namely, number of nuclei (one or two), number of tails (one or two), and colour (blue or red - as leading or trailing diagonals respectively) as in Figure 3.21.

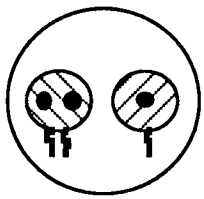


Figure 3.21 Cell Representation.

The instances can also be represented in propositional terms for the case: {(two, two, blue), (one, one, red)}, or in H^6 for unsupervised learning by the sixth dimensional boolean row vector [1,1,0,0,0,1].

The first term in each sublist equivalent stands for the number of nuclei, the second for the number of tails and the final term for the colour (red or blue). If this were to be a positive example of a cancerous cell then it would suggest the following rule:

if body1 of the cell has two nuclei, two tails and is blue,
and body2 of the cell has one nuclei, one tail and is red,
then the cell is cancerous.

The class of input is <attribute-value> where the input is a relational tuple table of positive and negative instances of cancerous cells. For supervised learning a further single instance boolean value 1 or 0 can be associated with the positive or negative classification respectively. The fact that each cell contains two bodies is of interest if the order of the bodies is unimportant, as shown, for example, in Figure 3.22.

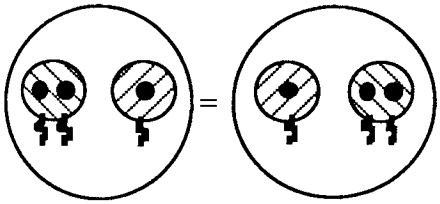


Figure 3.22 [[1,1,0,0,0,1],[0]] = [[0,0,1,1,1,0],[0]].

This results, generally (ie if no identity exists), in two nodes in quite separate parts of the hypercube having the same effect. The actual instance identifies only one of these nodes. Evidently we will end up with two totally (unless overlapping) separate (and in typical cases mutually exclusive), identical groups resulting in positive classification. This represents another example of capture by the hypercube of the important concept of *invariance*. Positive instances will cluster together into a connected group in the hypercube. This could be regarded as *noise* on an average group value. Parts of the group can then be seen as *continuous invariance*. Whereas, in the above case, the resulting separate groups can be seen as *discontinuous invariance*.

Generality ordering on concept descriptions presents two possible descriptions in the same format as the instances. Only some of the features are present. This indicates that the missing features are considered irrelevant. In the hypercube, a cluster or clusters of subhypercubes of positive instances excludes irrelevant features, that is, negative instance subhypercubes in other parts of the hypercube, and the resulting function is a description of the positive instance space. Suppose that hypothesis a) states that for a cell to predict cancer it must have one body with two nuclei and another body with one tail and a blue colour. If "?" denotes an irrelevant value then in propositional terms the description is {(two, ?, ?), (?, one, blue)}. This is represented in the hypercube by some lower dimension subhypercube. If hypothesis b) has one feature less, then b) is more general than a), that is, it covers more instances.

3.4.5.2 The structure of the space.

The partial ordering of the hypothesis space extends from the most specific to the most general. The most specific will correspond to actual instances since they are completely filled. The remainder of the space corresponds to hypotheses of varying levels of generality. In the hypercube this corresponds to subhypercubes of some subdimensionality. There is a partial ordering of the hypothesis space only, since hypotheses may be more specific than some other hypothesis but not more specific or more general than each other. It is this partial ordering that leads to a search through a sizeable hypercube lattice of potential concept description states in learning from examples. The generality dimension suggests two classes of operators for moving through this problem space.

1) Make an existing hypothesis more specific.

2) Make an existing hypothesis more general.

This *bias* suggests two basic focussing schemes for searching the space of concept descriptions. In case 1), begin with the most general hypothesis and as new instances are encountered, more specific descriptions are produced. In case 2), begin with the most specific hypothesis, moving to more general descriptions as more data is observed. Alternatively, it is possible to combine both directions, as in the version spaces family (Kodratoff, 1988) ***and in the subhypercube process*** - subhypercubes (generality end) reference node groups (specialisation end) for their classification in order to coalesce on a solution in the middle. A further possibility is to move towards more specific in some cases and towards more general at other times (middle out) - no such algorithm has had much success. Together the representation for states and the set of operators define a problem space. It is still necessary to search the resulting space in an effective manner or else the problems of searching in a high dimensional hypercube will result in a chronically slow algorithm.

3.4.5.3 The subhypercubing algorithm.

In section 3.4.4.3 we stated our algorithmic requirements. **The solution is a "subhypercubing algorithm"** as we will now show. This is an algorithm which searches the complete hypercube from H^n, at the top, down through the subhypercubes to parity, at the base. Hence it is "globally derived" (requirement 2, section 3.4.4.3). (Remember, we decided to account for the O set initially - since it is of the same type as the I set for any H^n, as discussed in sections 3.4.3, 3.4.4 and 3.4.4.1). The subhypercubing process has the advantage that it ultimately dissects out all of the possible functions within the complexity range. Since this occurs as a top down search it is "targeted" on the global function(s) (requirement 5, section 3.4.4.3).

Consider the H^2 subhypercube heterarchy given in Figure 3.23. At the zeroth subhypercube level is the hypercube itself or I, which for H^2 is a square. At the first subhypercube level there are by our "double rule" (in section 3.2.4) the 2n subhypercubes, which in this case are lines. At the second subhypercube level are the individual nodes. Careful examination of this heterarchy reveals only 5 types of function - one example of each type, depicted in the order in which they will be discovered by the subhypercubing algorithm, as given in Figure 3.24. This *accords with the **number** of function types* from the above cr theory (section 3.4.4.1). It also *accords in the **ordering** of the function types,* ie which ones will be discovered in which order. We assume, in theory at least, that subhypercubes of the same type being of the same level take approximately the same time to discover and are therefore of equal complexity.

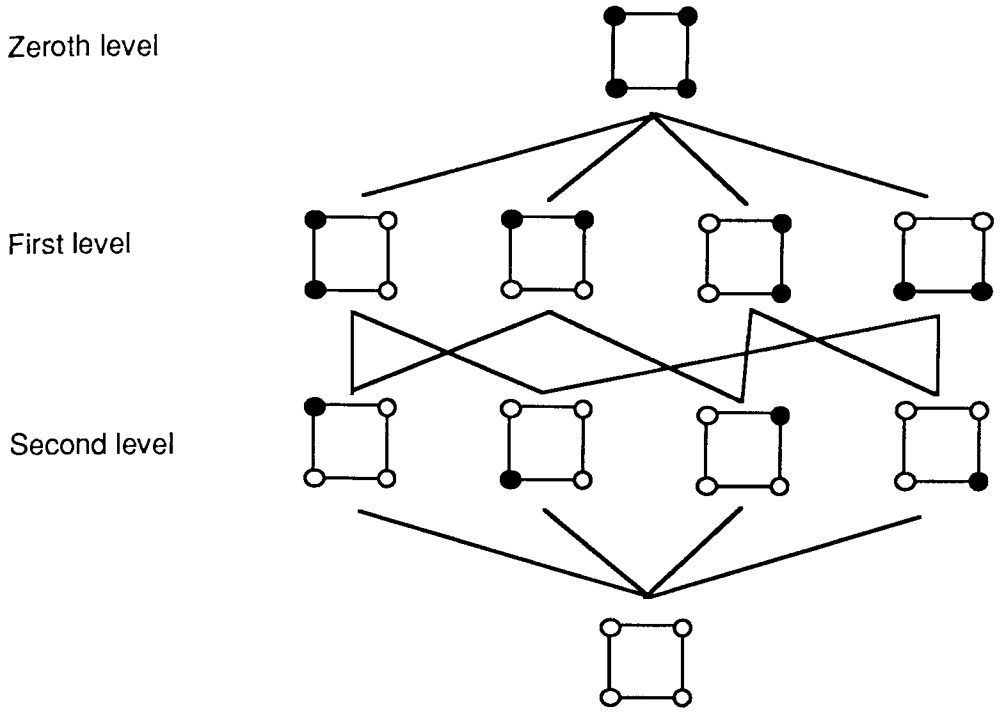


Figure 3.23 The H^2 subhypercube heterarchy.

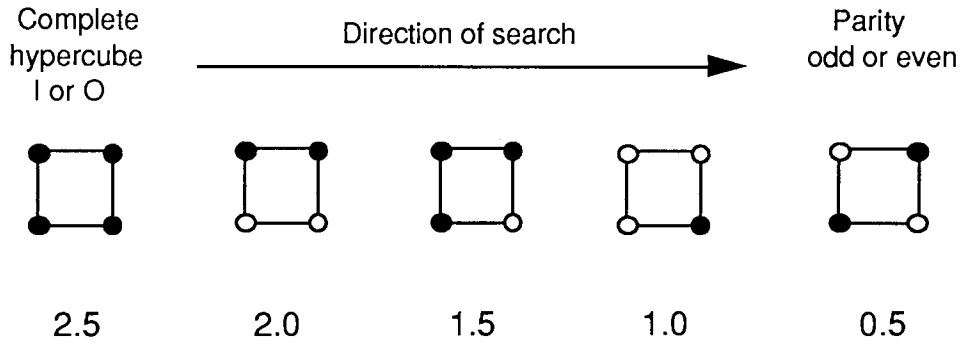


Figure 3.24 The five function types of H^2.

On the left hand side of Figure 3.24 is the complete hypercube, discovered first. On the right hand side is the parity function, discovered last. In between are the three remaining types of function for this dimension. This ordering can be labelled. The labelling symbol is arbitrary, **all that really matters is the ordering.** The simplest labelling is to put the 5 types in 1:1 correspondence with the numbers 1 to 5. In the case of cr the lowest number was associated with parity and the highest with the complete hypercube. If we use for the subhypercubing algorithm the same **convention for the search order** (start and end points the same) as for cr (section 3.4.4.1) then the result is in 1:1 correspondence with the cr range 0.5, 1.0, ..., 2.5. We can now fill in the column for the subhypercubing process, called "subh" in Table 3.5. Hence the ordering, search order, types of function and number of functions and therefore all individual functions in cr and subh are the same. We see that subh ≡ cr for all values. Since by the orthogonality of the hypercube there is nothing unique

about H^2 this argument will similarly apply for H^n. Hence we conclude the following. **Subhypercubing precisely satisfies the complexity range criterion.** Thus subhypercubing is in 1 : 1 correspondence with cr (requirement 4, section 3.4.4.3).

The subhypercubing process can enormously cut down on the computation required (requirement 6, section 3.4.4.3) since in order to test if subhypercube 's', say, is complete we only require one example that 's' is *not* complete whatever the size of 's' in number of nodes. If 's' is not complete then it can then be partitioned into its own first level subhypercubes. Therefore we satisfy the ϕ-boolean (section 3.4.3.2) need (requirement 9, section 3.4.4.3). Typically there is no need to test each and every node as with cr. One way of looking at this is that it follows from basic geometrical considerations that **an alternative principle to selecting nodes** (cr driven) to represent the minimum polynomials (join of meets) of the simplest possible solution **is to select faces and higher structures** (hence subhypercube driven), again by minimum polynomials. Since subhypercubing is globally driven and produces minimum polynomials it satisfies the necessity (section 3.3.3) of finding the simplest possible solution (requirement 1, section 3.4.4.3).

Many further algorithmic refinements are possible to fight the exponential explosion with increasing dimensionality. For example, associative memory can be created by use of the complement. The complement of any function is given by the join of those nodes *not* incorporated in the function. Sometimes it is more convenient to use the complement form eg the opposite specification to that required may provide the easier route, by then transferring the result directly to the complement. The important property that adjacent nodes differ in respect of only one input element state permits a minimisation process. Our symmetry concept is any 1:1 transformation which leaves the distances between all pairs of nodes unaltered. There are $2^n n!$ symmetries in the hypercube: for example, rotative symmetries, reflective symmetries, identity symmetries, etc. H^3 has 48 such symmetries and H^4 has 384 symmetries. Various such insights can provide algorithmic refinements. These points will be discussed further in the next chapter.

The subhypercube process requires an associative hypercube for referential consistency. This automatically satisfies the preference for an associative hypercube (requirement 10, section 3.4.4.3). Finding by a global search process the precise set of subhypercubes that represent the universe is analogous to finding by local iteration the precise set of weights that represent the global minimum in more conventional systems. In both cases the result may be approximate to a specified level of confidence or an exact match. With a subhypercubing algorithm one huge advantage is that we may specify the depth of search by levels (Figure 3.23). By the "double rule" (section 3.2.4), if the hypercube is of dimension 50 then there will be 100 first level subhypercubes. This already partitions the space into 1 part in 100. Rarely does one need an error of less than 1%. Nevertheless, we can always run

to the 2nd, 3rd, ... level as required. Such **"resolution" has a remarkable advantage, it is not NP** but monomial and slowly increasing arithmetically! The equation is of the form $t = nc_1 + c_2$ where t is the time taken, c_1 and c_2 are constants and n is the dimensionality, as in H^n. This satisfies the need for resolution (requirement 11, section 3.4.4.3) and makes a huge impact on the speed of the algorithm for large problems (requirement 6, section 3.4.4.3).

Given n in H^n, the "double rule" (section 3.2.4) can calculate the connectivity of all first level subhypercubes. It is, therefore, unnecessary to record them as such, so we satisfy the need for a largely virtual hypercube (requirement 7, section 3.4.4.3). Typically, we would expect the hypercube to be exceptionally sparse, so the saving is not only in memory but also a further speedup in computation time (requirement 6, section 3.4.4.3). On the other hand, we must record the specific values of exemplars and their associated classification for use by the algorithm, otherwise it has nothing to work on! This satisfies the need for actual exemplar storage (requirement 8, section 3.4.4.3). These exemplars will have a classification of 0 or 1. If the subhypercubing process finds no contrary classification for that subhypercube then it will classify that subhypercube as 0 or 1 in accordance with the exemplar. This implies that the algorithm is "balanced" (requirement 3, section 3.4.4.3) as far as the exemplars are concerned.

We have satisfied all 11 requirements in section 3.4.4.3 for the required algorithm. We are, therefore, prepared to temporarily overlook *one* grey area. This concerns the balanced approach to *virtual* storage connections. No classification is available. How should this be handled? A random assignment of 0 or 1 could give severe distortions. In practice, we feel that the hypercube will be very sparse. (For example, if the dimensionality is 20 and we have 80 exemplars to get a reasonable distribution, then there are only 80 exemplars amongst about 10^6 nodes, which is very sparse indeed). Our solution is to assume, in the absence of evidence to the contrary, that all virtual connections are of class 0. Analogously, in learning the concept "tree" we should not assume very different objects, eg "bananas" are also trees unless told otherwise. It remains to be seen just how distorting this simplification is. Formally, given a concept of attributes A we assume all A´ are not A. This is reasonable, given the evidence. However, if $A \cup A' \neq \Theta$ because they form a subset of Θ, then we still assume that the grey area in-between, about which we may know nothing, is still A´, that is, not part of some future extension of the concept A.

3.4.5.4 Theoretical Predictions.

The subhypercubing algorithm has been theoretically shown to satisfy the 11 requirements (excepting our "grey area") in section 3.4.5.3. These requirements also act as advantages of the program behaviour of our enhanced subhypercubing algorithm. We should

attempt to programme this theory, as closely as possible, into a machine learning program so that the program behaviour *is* the theoretical predictions (section 3.1.3). Therefore, our **theoretical predictions are precisely the same as the 11 requirements.** If a practical implementation of the subhypercubing algorithm can be programmed then testing the validity of these advantages in practice would form an excellent series of practical experiments due to their solid links into the theory (section 3.1.3).

3.4.5.5 The hypercube represents the universal set.

The complete hypercube in all possible variations, since it represents all possible combinations within the dimensionality of any given problem, also represents the **universal set**, Θ, for that problem within the scope of its stated specification. This is in contrast to other formalisms, such as many types of neural net, where, depending upon the initial values of certain parameters, only some *subset* of the universal set may be considered; thereby perhaps leading to the learning of some local minimum potential energy rather than the *global* minimum.

It is interesting and instructive to compare the subhypercube algorithm which is global in action with the commonest neural net machine, namely "error back propagation" or "back propagation." Back propagation, like almost *all* learning algorithms, is a hill-climbing algorithm and hill-climbers are **local** algorithms. The ever present danger with a local algorithm is that, since it 'searches' locally, only a local minimum may at times be discovered, as represented in Figure 3.25 (b). (Note: that in Figure 3.25 we alternatively refer (b) to a potential well rather than hill. The surface is also possibly, typically much flatter than illustrated here).

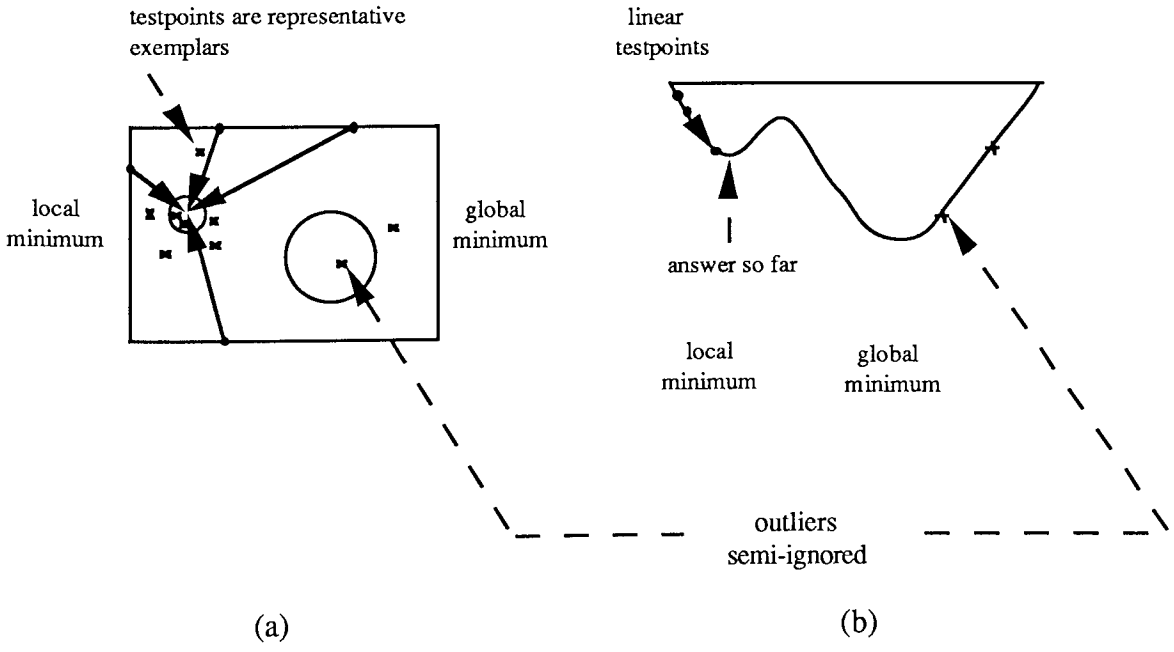


Figure 3.25. The local action of back propogation as a hill-climber.

Different parameter settings have the effect of choosing different starting points, but it may still only reach the local minimum - small circle in Figure 3.25 (a) - and then even "get stuck." In the example given in Figure 3.25 (a), many of the exemplars are near the local minimum and outliers are therefore semi-ignored (the "herd effect" all attack same problem, ignoring all others). The algorithm may be slow or possibly never actually reach the global minimum because it is converging locally by comparing weights in what may be an unrepresentative space with respect to the universal set. That is, convergence to zero error for the exemplar surface is the only one which back propagation can know about. This is entirely different from zero *universal* error, which could even still be, say, 99 %.

The hypercube, on the other hand, is a (much rarer) global algorithm, assuming a subhypercubing algorithmic search. Unlike back propagation it does not merely compare nodes. In contrast, it searches for global symmetry by universally iteratively and orthogonally inserting decision surfaces in increasing detail until sufficient to separate the positive and negative exemplars. This process is illustrated pictorially only in Figure 3.26. Although the well-known problems of hill-climbers such as ravines are avoided, there is however no panacea in a global algorithm, since there is an inherently greater susceptibility to noise unless care is taken.

The ever present danger with a global algorithm is that, since it searches globally, undetected noise may perturbate the result. This is because outliers are *not* ignored. The global algorithm may, therefore, be faster reaching the approximate global minimum because it is converging globally even in what may well be an unrepresentative space with respect to the universal set - but with totally unrepresentative exemplars it may well, of course, still fail to reach the global minimum.

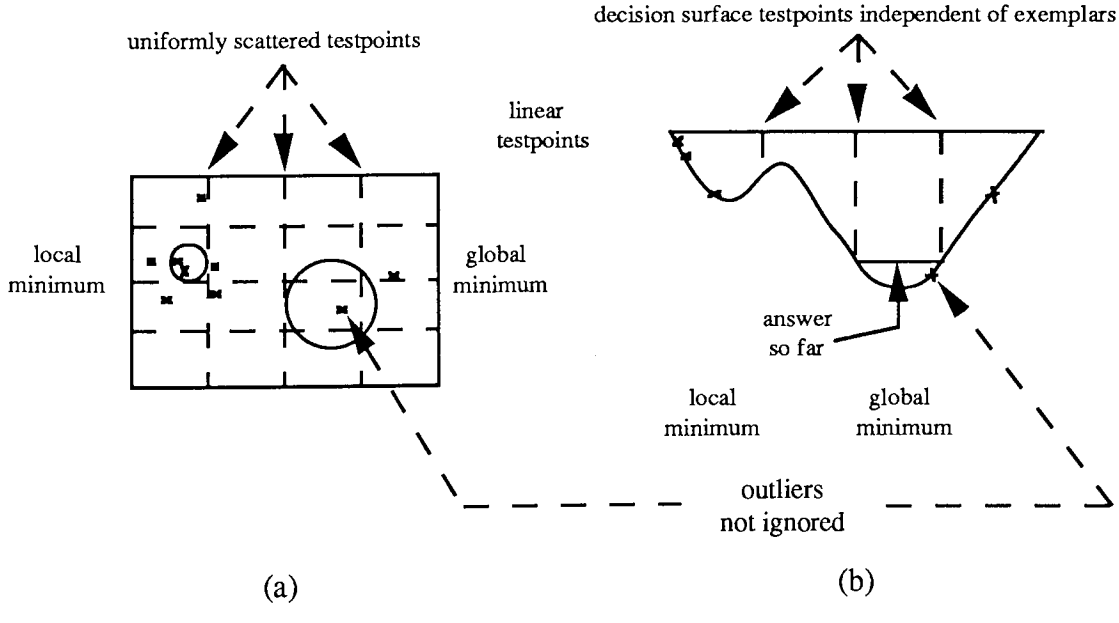


Figure 3.26 The global action of the hypercube.

The comparison can be summarised as follows: in practice back propagation iterates nodewise by altering synaptic weights. Therefore its resulting space can only effectively span the representative space of the given exemplars and in a sparse and poorly representative space a new and unseen example from the universe may be assigned incorrectly due to the **under-representation of outliers** - see the dotted line in Figure 3.27.

Whereas the hypercube, searching globally, iterates uniformly and sub-globally viewing all nodes as representative of their particular universal subspace and hence the hypercube takes account of any unseen example from the universe in a way that is much more likely to be correct - albeit at the possible cost of **outliers being taken as over-representative** in the case of noise.

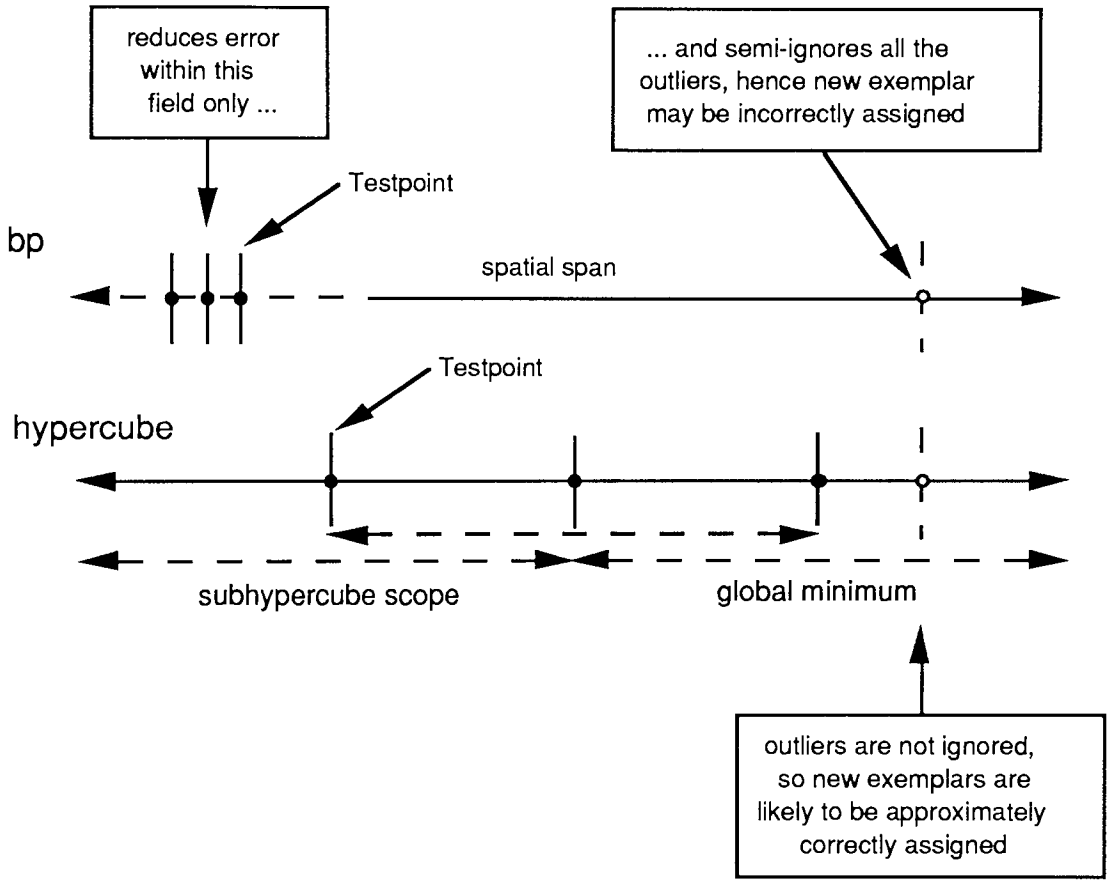


Figure 3.27. The effect of testpoints in bp and the hypercube.

3.4.6 Theoretical scoping.

We now discuss the scoping aspects of the theory referred to in section 3.4. We repeat our approach to scoping as stated in section 3.4. Our stated purpose is served in the following way: for the most part we are content with iteratively defining and then combining terms into more complex structures using standard pure mathematics for later examination of their properties with respect to the hypercube. A standard notation is used throughout and listed for reference in Appendix 1. Mathematical definitions are also given in Appendix 2.

3.4.6.1 Relations.

The elements of the set or sets then have certain relations between them. We can define various associated concepts. A binary relation $\Re$ between sets A and B assigns: $\forall \langle \alpha, \beta \rangle \in A \otimes B \exists \wp (\alpha \Re \beta \vartheta \alpha \Re' \beta)$.

We can call the solution set of relation $\Re$ between sets A and B the set of all ordered pairs: $\langle \alpha, \beta \rangle \in A \otimes B \ni \alpha \Re \beta$

We can define a number of basic types of relation as follows:
A relation $\Re$ on set A is reflexive if: $\alpha \Re \alpha \forall \alpha \in A$.
The relation $\Re$ on set A is symmetric if: $\alpha \Re \beta \forall \beta \Re \alpha \ni \alpha, \beta \in A$.
The relation $\Re$ on set A is antisymmetric if: $\alpha \Re \beta \wedge \beta \Re \alpha \Rightarrow \alpha = \beta \ni \alpha, \beta \in A$.
A relation $\Re$ on set A is transitive if: $\alpha \Re \gamma \forall (\alpha \Re \beta \wedge \beta \Re \gamma) \ni \alpha, \beta, \gamma \in A$.

A partition ρ of set A separates the elements α into subsets: $\ni \alpha \wp \rho \forall \alpha \in A$.
The quotient set ε of a set with an equivalence relation defined on it is the set of all equivalence classes. A relation $\Re$ on set A which is reflexive, symmetric and transitive $\ni \Re$ partitions A into non-overlapping subsets of: $\varepsilon \ni \alpha \wp \varepsilon \forall \alpha \in A$.

We define the equivalence class of set A as a subset belonging to a partition of A by an equivalence relation. Ordering is an important concept for hypercubes as we have seen. We define an ordered set as a set with an order relation defined on it. In particular we are interested in partial ordering.

A **partial order relation** is a relation on a set which is reflexive, antisymmetric and transitive. Whereas a total order relation between sets A and B is a partial order relation $\Re$ on A: $\ni \forall \langle \alpha, \beta \rangle \in A \otimes B \ni \alpha \neq \beta \exists \wp (\alpha \Re \beta \vartheta \beta \Re \alpha)$.
The converse relation ® of a given relation $\Re$ between sets A and B is the set of all reversely ordered solution set ordered pairs: $\langle \beta, \alpha \rangle \in B \otimes A \ni \beta ® \alpha \forall \langle \alpha, \beta \rangle \in A \otimes B \ni \alpha \Re \beta$.

3.4.6.2 Bounds.

The concept of bounds is basic to the hypercube. A lower bound of subset B of a partially ordered set A is any element: $\alpha \in A \ni \alpha \Xi \beta \forall \beta \in B$ An upper bound of subset B of a partially ordered set A is any element: $\alpha \in A \ni \beta \Xi \alpha \forall \beta \in B$ This leads to the following basic lattice concepts:

The **greatest lower bound** (glb) of subset B of a partially ordered set A, the infimum of B written as inf B, is that lower bound α where $\alpha \in B$ for every lower bound β of B: $\ni \beta \Xi \alpha \forall \beta \in B$.

The **least upper bound** (lub) of subset B of a partially ordered set A, the supremum of B written as sup B, is that upper bound α where $\alpha \in B$ for every upper bound β of B: $\ni \alpha \, \Xi \, \beta \, \forall \, \beta \in B$.

3.4.6.3 Operations.

We stated that the elements of a set may be compared and combined resulting in certain relations between them. But it is possible to perform operations on sets. That is combine individual elements to form new objects. We can again define various associated concepts.

A binary operation on set A assigns a unique element β to all ordered pairs of elements $\langle \alpha, \alpha \rangle \in A \otimes A$ and the set A is closed under this operation iff $\forall \, \beta \in A$.
A closed binary operation $\Im$, on set A $\Rightarrow$ set A is closed under the operation $\Im$, iff set A assigns a unique element $\beta \, \forall \, \beta \in A$ to all ordered pairs of elements $\langle \alpha, \alpha \rangle \in A \otimes A$.
A commutative binary operation $\Im$, on set A has: $\alpha \, \Im \, \beta = \beta \, \Im \, \alpha \, \forall \, \alpha, \beta \in A$
An associative binary operation is a closed binary operation $\Im$ on set A:

$$\ni (\alpha \, \Im \, \beta) \, \Im \, \gamma = \alpha \, \Im \, (\beta \, \Im \, \gamma) \, \forall \, \alpha, \beta, \gamma \in A.$$

A distributive binary operation is a closed commutative binary operation $\Im$ on set A which is said to be distributive over a closed binary operation $\aleph$ on A iff:

$$\alpha \, \Im \, (\beta \, \aleph \, \gamma) = (\alpha \, \Im \, \beta) \, \aleph \, (\alpha \, \Im \, \gamma) \, \forall \, \alpha, \beta, \gamma \in A.$$

The identity element e for the binary operation $\Im$ on set A is:

$$e \, \wp \, A \ni e \, \Im \, \alpha = \alpha \, \Im \, e \, \forall \, \alpha \in A.$$

In a certain, sense relations and operations are two sides of the same coin - hence the hypercube lattice is able to simulate these operations.

3.4.6.4 Lattices.

We are now in a position to define a **lattice** as a set with a partial order (poset) relation defined on it such that any subset of two elements has both a greatest lower bound, called the infimum and a least upper bound, called the supremum.

Let A be a subset of an ordered set S. The supremum of A, sup A, is an element of S such that sup A is an upper bound of A. If c is any upper bound of A then sup A $\leq$ c. Similarly, the infimum of A or inf A is an element of S such that inf A is a lower bound of A and if d is any lower bound of A then d $\leq$ inf A.

We define a lattice as a set with two operations, satisfying certain propositions. The partial ordering and "inf" and "sup" properties of a lattice are equivalent to the following properties of a set with two closed operations, $\Im$ and $\aleph$.

1) $\alpha \Im \alpha = \alpha, \quad \alpha \aleph \alpha = \alpha.$

2) $\Im$ and $\aleph$ are commutative, eg

$$\alpha \Im \beta = \beta \Im \alpha, \quad \alpha \aleph \beta = \beta \aleph \alpha, \quad \forall(\alpha, \beta) \in A.$$

3) $\Im$ and $\aleph$ are associative, eg

$$(\alpha \Im \beta) \Im \gamma = \alpha \Im (\beta \Im \gamma) \text{ and}$$
$$(\alpha \aleph \beta) \aleph \gamma = \alpha \aleph (\beta \aleph \gamma), \qquad \forall(\alpha, \beta, \gamma) \in A.$$

4) $\Im$ and $\aleph$ are absorbtive, eg

$$(\alpha \Im \beta) \aleph \alpha = \alpha \text{ and}$$
$$(\alpha \aleph \beta) \Im \alpha = \alpha, \qquad \forall(\alpha, \beta) \in A.$$

If we associate $\Im$ and $\aleph$ with inf and sup by

$$\alpha \Im \beta = \inf\{\alpha, \beta\}; \quad \alpha \aleph \beta = \sup\{\alpha, \beta\},$$

then these four properties follow readily from the properties of inf and sup. Lattices of sets find applications in logic, as below, when the operations are associated with the meet and join of sets in the hypercube representation of functions.

3.4.6.5 Structures.

We are now in a position to see the real scope of possibilities for the hypercube. We define a mathematical structure as a set with one or more operations and/or relations defined on it. At the simplest level a groupoid is a set with a closed binary operation defined on it. At the next level a semigroup is a set with a closed associative binary operation defined on it. A monoid is a semigroup with an identity. With this much structure the concept begins to be very useful. There are many practical applications of group theory.

We define a **group** as a monoid with the additional property given identity e, an inverse α^{-1} and binary operation $\Im$ that: $\forall \alpha \in A \; \exists \alpha^{-1} \ni \alpha \Im \alpha^{-1} = \alpha^{-1} \Im \alpha = e$
Further additions are a commutative group which is a group which has a commutative binary operation and rings.

A **ring** $(A, \Im, \aleph)$ is a set A with two binary operations $\Im$, $\aleph$ defined on it such that $(A, \Im)$ is a commutative group, $(A, \aleph)$ is a semigroup and $\aleph$ is distributive over $\Im$.
Another refinement is an ideal which is a subring $(B, \Im, \aleph)$ of a ring $(A, \Im, \aleph)$ with the property that $\alpha \Im \beta \in B$ and $\beta \Im \alpha \in B \; \forall \alpha \in A, \beta \in B$.

Another separate concept is that of a **field** which is a commutative ring $(A, \Im, \aleph)$ with an identity and the further property that, excepting the identity for $\Im$, every element has an inverse with respect to $\aleph$.

3.4.6.6 Higher structures.

Structures of a more complex kind are possible by having multiple sets with relations and operations defined on them. The concept of a correspondence is basic. A correspondence from the codomain of set B relates each element of the domain A to one or more elements of B. Another useful concept is that of an image. The image of element α of the domain is the set of all elements of the codomain of a correspondence which are related to α.

Various types of correspondence are possible. An onto correspondence is a correspondence such that each element of the codomain is an image of at least one element of the domain, the image of the domain being the complete codomain. An into correspondence is a correspondence such that the image of the domain is a proper subset of the codomain.

A one-to-one correspondence has a unique image for each element of the domain and no element of the codomain is the image of more than one element of the domain.

A many-to-one correspondence has a unique image for each element of the domain and at least one element of the codomain is the image of more than one element of the domain.

A one-to-many correspondence has a unique image for each element of the domain comprising of more than one element of the domain and each element of the codomain is the image of a unique element of the domain.

A many-to-many correspondence has a unique image for at least one element of the domain comprising of more than one element of the codomain and at least one element of the codomain is the image of more than one element of the domain.

The concept of a function is basic to higher mathematics and essential to science and engineering. A function is a correspondence which is either one-to-one or many-to-one. Subtypes exist. An injection is a one-to-one function. A surjection is an onto function. The inverse function is the converse of a bijection.

Lastly, we consider the concept of a morphism and define a morphism as a function from (A, $\Im$) to (B, $\aleph$) such that the structure (B, $\aleph$) models the structure of (A, $\Im$) and hence $\Rightarrow$ that the image of $\alpha \Im \beta$ equals the combined images of α and β under $\aleph$ $\forall \alpha, \beta \in$ A. The kernel of a morphism is that subset of the domain of a morphism such that the image is the identity of the codomain. Isomorphic refers to a morphism which is one to one and onto, hence a bijection.

By simulating operations as relations we have indicated that the hypercube is able to scope a number of important (and considerably sized) mathematical subjects including, at

least, functions, groups, rings, fields and morphisms; but we have not proved that all possible functions etc. are within scope. Time has prohibited further analysis.

3.5 LEARNING.

The subhypercubing algorithm provides a basis for a basic inductive *learning* machine whose theoretical scope is *at least* that theoretically outlined above (in sections 3.4.6 to 3.4.6.5). The subject of learning poses some very difficult problems. Fair and continuing progress has been made with the subject but the boundaries of what is known still contain fuzzy areas. Problems are discovered which were previously ignored or unrecognised (the "hidden variables" referred to in section 3.1). Inadequacy of language can be a source of such problems. Therefore let us consider the word **"learning"** and in particular with respect to the hypercube.

What exactly do we mean by the term "learning?" In the following pages we attempt to dissect this concept in order to understand some of the capabilities and limitations of the hypercube as a learning machine.

3.5.1 Bias, preconceived ideas and domain specific knowledge.

As we have seen **bias** is very important. According to the bias the choice is made. We considered bias in some detail in section 2.2.3.2. Without bias, induction is impossible. With bias, the choice is made according to the bias. The various possibilities, therefore, represent the various possible biases. This is of the utmost importance with considerable implications. We consider the matter in depth, as follows:-

A system lacking inductive bias would not, **a priori,** be capable of making predictions (ie choosing reliably) *beyond the training examples it had already seen!* The crucial point being that were a system to have the *advantage* of having "no preconceived ideas" it would then also have the *disadvantage* of the lack of an inductive basis upon which to make predictions beyond the training examples already seen. This is the extreme empirical case.

Reversely, the system is able to make predictions on the basis of generalisations from the examples given. In order to do this "preconceived ideas" are *necessary* (not necessarily domain dependent, they may usefully be domain independent). Yet the preconceived ideas may require prior domain knowledge, as we saw with EBL machines (section 2.2.3.1). The crux of the matter is that both tautologically and paradoxically, in the extreme deductive case, the system must already completely contain that domain specific knowledge which it is seeking to gain (section 2.2.3.5).The extent of the overlap with its

recursive implications determines the practical viability of the system. It follows that the extent of the tautology **necessarily** limits the system's ability to learn - which is, of course, **exactly** the problem that the system is trying to address!

Our solution is to take the middle ground on the following basis.

We do not want the system to lack inductive bias - the tabula rasa approach achieved nothing and is the height of arrogance. Why should it necessarily be feasible for us to produce a machine which starts off knowing absolutely nothing, and after being left by itself for a while, ends up with an intelligence? Such is our arrogance.

Conversely, neither can we countenance it as progress to "assume the proof" with respect to pre-programmed domain specific knowledge. To do so would be to eliminate the necessity to learn that same domain specific knowledge. In what sense can we then call it *learning*?

Thus, we see the middle ground as having the only possibility for maximal learning transfer, with the ex*tremes on eith*er side representing minimal learning. This brings to mind a bell-shaped or typical normal distribution curve, as Figure 3.28.

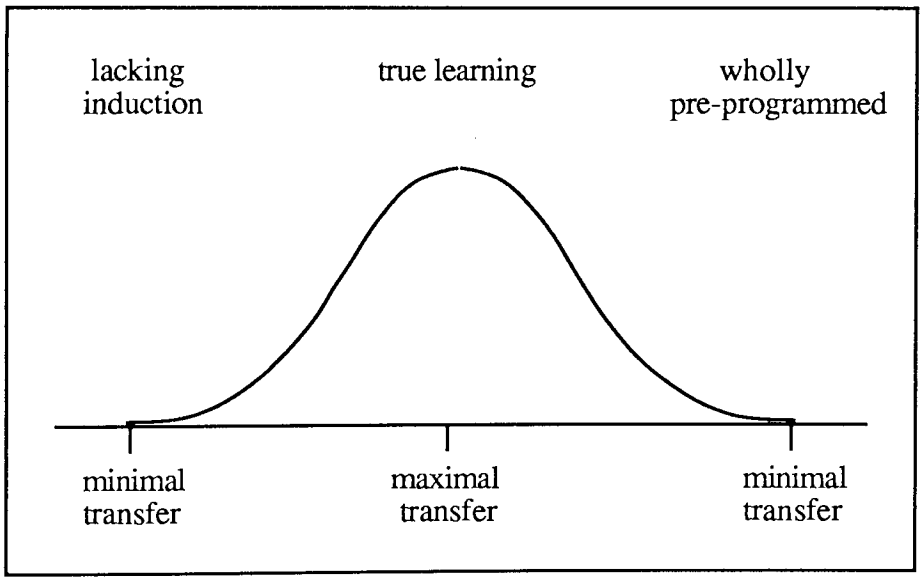


Figure 3.28 The preconceived learning curve.

3.5.2 Domain independence.

What precisely do we mean by the term middle ground? As discussed in section 3.5.1, preconceived ideas are essential for prediction purposes. Further, we should attempt to avoid the above recursive problem of domain knowledge. The worked example in section

3.3.2 hints at a reasonable solution. In the worked example specialised domain knowledge was not considered. Rather, we considered the implications and effectiveness of various general purpose learning algorithms. That is, metaphorically, we are asking the question: "How far down the road will it get us if the machine were to utilize a **domain independent general purpose learning algorithm?**"

3.5.3 Low level learning.

In looking around for clues to the possible chances of success we might consider learning in the simplest animals which have a definable nervous system. [11] If we restrict the case to considering a low level general purpose learning machine, then a domain independent algorithmic approach may have merit. If the foregoing assumption is reasonable then it implies that the suggested domain independent algorithm for low level problems of even considerable size may have useful areas of application without there being the necessity to invoke, **a priori**, domain specific knowledge.

3.5.4 Top-down and bottom-up learning.

How far can we take this argument? At what point does the top-down effect of domain specific knowledge count? The answer is unknown and is a hot topic in many branches of AI. For example, David Marr, one of the great pioneers of vision, assumed that the answer is *never* (Roth and Frisby, 1986); but there are arguments on both sides. Top-down processing provides a guide, the constraints of which enormously simplify processing; but if the expectation is incorrect then the wrong thing is learnt. Examples of this may be "getting the wrong end of the stick" and visual illusions. Bottom-up processing makes no assumptions about what is there; but the combinatorial possibilities may mean that the global picture could well be missed. An example might be "failing to see the wood for the trees."

These difficulties are at least to some extent avoidable. The reader may have noticed that we coupled together "domain specific" and "top-down" in the usual manner. Nevertheless, they actually represent separate problems. Domain specific knowledge can be

---

11 We assume that it is unlikely that neural systems containing up to a few dozen neurons (barely beyond amoeba) would contain *unlearned* domain specific knowledge, certainly in the sense we understand it. Rather, we suspect that a domain independent learning mechanism may well suffice. Extending this argument to at least the first few neural levels of any modality again seems reasonable. For humans this implies that we are considering systems comprising millions of neurons. Yet even at this level of complexity, it would seem ridiculous to assume, for example, that the first five neural levels comprising the retina utilize *unlearned* domain specific knowledge. Surprisingly, even rods and cones in the retina do some learning. Yet it is surely unreasonable to assume that some given rod or some given retinal neuron "knows" that it is, say, looking at a tree - we would be left with the problem of searching for a brain inside each neuron! It is surely even more unfounded to assume that this given rod already "knows" about trees *prior* to its first use! Yet such neurons do learn. Hence, it seems reasonable to postulate a domain independent learning mechanism.

learnt as opposed to being a priori knowledge. Marr's preference was for this learnt knowledge (Roth and Frisby, 1986). He questioned whether there is a necessity to invoke the top-down concept in order to explain vision and concluded that the answer was "no."

3.5.5 Domain independent top-down focusing.

Again, the use of domain specific knowledge in the top-down guidance of processing is typical but not the only way. Domain independent processing used in a feedback manner from the higher levels to focus lower level processing offers an alternative which would also seem to take much of the steam out of the above coupling. We have not pursued this enhancement, since it implies a considerable extension to the mainstream of the present work. Nevertheless, it is possible to conceive a multiple hypercube structure where certain 'higher level' processing hypercubes act as feedback paths in the above manner.

3.5.6 Symbolic and non-symbolic learning.

As argued, it is best to restrict application of the learning machine to low level learning. This we see as solid ground. It is simple enough to be a realistic aim. For example, we avoid the brittleness problems of natural language processing, referred to earlier in section 3.1.1. Our reasoning is as follows:

Words are symbols. Where the meaning of symbols is obtained merely by reference to other such symbols, no semantic understanding can follow, only syntactic correctness.

For example, if the machine uses the word "you" in the course of its dialogue with the user, it will have almost zero appreciation of what that means relative to our understanding of the term. There is unlikely to be any referencing of the symbol "you" to semantics, because the syntactic/semantic divide precisely delineates the state of the art in natural language. Whether reference to non-symbolic modalities, such as, in this case, an internal image of the user, would help much is presently unanswerable. The implied symbolic/non-symbolic interface is an area of considerable interest to current researchers in many fields - and a highlight of this research. It is not surprising that current systems produce the typical problems of AI such as brittleness and idiot savant behaviour.

3.5.7 Defining low and high level learning.

The constraints of low level and domain independence raise *two major points.* Firstly, up to now we have been using the terms 'high level' and 'low level' without definition. We have delayed any definitions until these terms can be more meaningfully

discussed. [12] From the above discussion we extract three major boolean variables of interest, and *assume them to be of equal importance,* as below.

1.	non-symbolic,	symbolic,
2.	domain specific,	domain independent,
3.	bottom-up,	top-down.

There are eight possible combinations of these triplets such as:

< top-down, domain independent, symbolic >

The terms 'high level' and 'low level' when applied to ourselves are, as we saw above, opaque. The same terms, when applied to conventional computers are simply assigned according to the nearness of a computer programming language to human language or machine language, respectively.

We are interested in looking at these terms with reference to an *intermediate situation* between opaque complexity and a simple boolean choice. The "intermediate situation" concerns the defining of the terms high level and low level for purposes of *machine learning.*

It is required that this task be done adequately enough to at least minimally encode the range of possible simple learning applications into these terms sufficient to differentiate monotonically increasing sub-levels of difficulty. Hence these definitions will refer only to the machine learning world.

It is easy to see that we can define low level as:

< bottom-up, domain independent, non-symbolic > [13]

Similarly, it is easy to see that we can define high level as:

< top-down, domain specific, symbolic > [14]

This leaves six remaining triples without type. In seeking a comparison with our human experience of learning we see that our use of language fails us and our words appear

---

12 Note: that it is of little use to make the distinction based upon the human experience of the conscious and the subconscious - since we have no processing model for consciousness. We therefore try another tack.

13 For example, projected machine learning situations analogous to the hypercolumn learning of "bars" (lines at specific angles) in the striate cortex, Hubel and Weisel, (1978).

14 For example, projected machine learning situations analogous to our human learning of, say, an academic textbook.

to be fuzzy and inadequate. There is this region in the middle for which we have no specific word or words, as in Table 3.7. [15]

	DS	dS	Ds	ds
T	H	?	?	?
t	?	?	?	L

KEY

S, s	symbolic, nonsymbolic
D, d	domain specific/independent
T, t	top-down, bottom-up
H	high level
L	low level
?	unknown level

Table 3.7 Triple matrix: high to low level.

For reference to individual cells within the matrix, the matrix triple format can be used.

e.g. < TDS > is one and the same as H or high level.

All further progress appears to be thwarted. This leaves us in the difficult position of having *no complexity metric* to guide us. For example, how many sub-levels are there: 2, 3, 4, 5 or 6? How are these sub-levels related? Is **< T, d, s >** above, below, or at the same level of complexity as, say, **< t, D, S >** and why? In such circumstances, the only safe bet would be to restrict the hypercube solely to low level problems as already discussed. The only escape from this trap appears to be the consideration of appropriate morphisms (section 3.4.6.6).

3.5.7.1 The machine learning morphism complexity metric.

Morphisms and isomorphisms are defined in Appendix 2. A good way to visualize the mathematical concept of morphism is to consider the relationship between a complex object and its corresponding simplification. It is an important part of the concept of morphism that the morphism captures and structurally preserves the essential components of the original object within its image. A good analogy is the relationship between one's self and one's shadow. The detail is missing from the image but the basic outline is there. The concept goes far beyond mere analogy - closer, in fact, to a precis.

Consider the morphism of the use of English as against basic neural processing relative to its image successfully exemplified in the high and low level languages of serial

15 Possibly this accords with our experience of consciousness and thought at the highest levels and awareness of the external world, sense impressions at the lowest levels and considerable oblivion of what happens in the huge sub-conscious area in-between!

machines. This morphism has proved to be a powerful and useful concept and led to the widespread use of the serial machine as Figure 3.29.

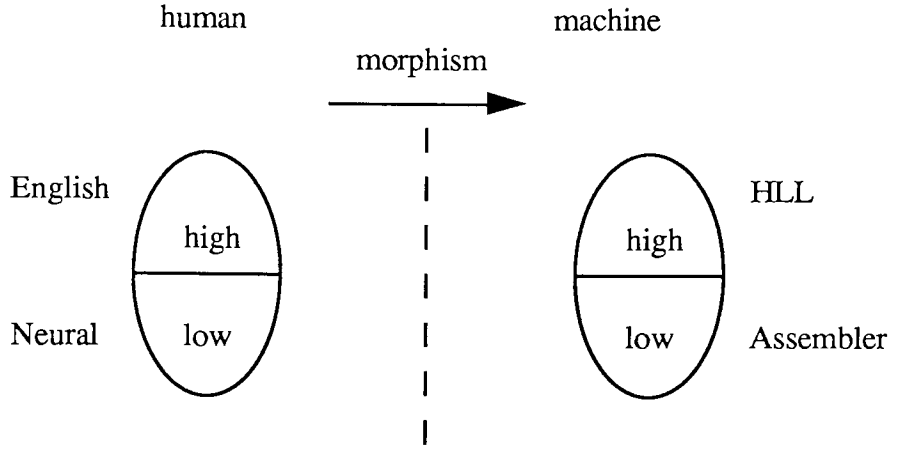


Figure 3.29 The serial machine morphism.

Morphisms are also of considerable interest in pure mathematics. Morphisms are found to be extremely useful tools for investigating the properties of the *original object*. The purpose of introducing this concept is to propose an analogous basic morphism for machine learning, as illustrated in Figure 3.30.

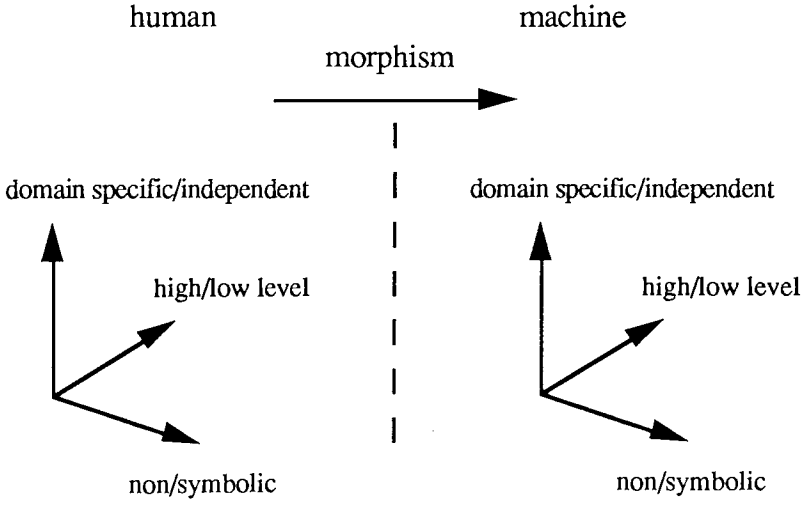


Figure 3.30 The machine learning morphism.

Clearly, if we are to try to attempt to emulate the success of the serial machine morphism, then the idea must be to attempt to capture the essential *image* of human learning abilities, in an equally powerful and useful way in a machine learning engine.

3.5.7.2 Sub-level complexity relationships.

The "intermediate situation," referred to in section 3.5.7, we nevertheless defined for the machine world. The crucial point being to consider the morphisms in the "intermediate

situation" between opaque complexity and the simple boolean choice, as in section 3.5.7. This is helped by:

a) Assuming that these morphisms have a *unique* solution for the "intermediate situation." This constrains a match or at least a mesh.

b) Unfortunately, it is hardly likely that these morphisms are isomorphic (certainly not the upper one). Note that we have assumed in section 3.5.7 that the triplet variables are equally weighted in importance. Hence we have *symmetry*.

Given the symmetry and the morphisms, Table 3.7 reduces to Table 3.8. Note the interlocking "ell" shapes.

	DS	dS	Ds	ds
T	H	h	h	l
t	h	l	l	L

KEY

S, s	symbolic, nonsymbolic
D, d	domain specific/independent
T, t	top-down, bottom-up
H, h	high level, sub-high level
L, l	low level, sup-low level

Table 3.8. Triple matrix: sublevels

The equal weighting enormously simplifies the problem. It follows that there are four sub-levels: H, L and two intermediate sub-levels referred to as h and l in Table 3.8 as sub-high level and sup-low level (where sup stands for "superior"), respectively.

Complexity change is dependent on going from low level to high level or vice versa, and is independent of triplets at the same sub-level. Given all this, there is only one solution, namely, each level change alters precisely *one* of the triple variables.

e.g. An example pathway from L to H is given by:

< t , d, s > -> < T, d, s > -> < T, D, s > -> < T, D, S >

The cells in the triple matrix, Table 3.8, are related complexity-wise in a uniform manner reflecting their equal importance. This is illustrated in Figure 3.31 where the arcs define cell conservation. The symmetry is now clear and each triple state helps to define a complexity pathway. Figure 3.31 beautifully defines the sub-level complexity relationships as a H^2 hypercube in TDS states. In the above example, TDs -> TDS was not apparent from Table 3.8 since it jumps a cell, whereas it is clearly in the pathway in Figure 3.31.

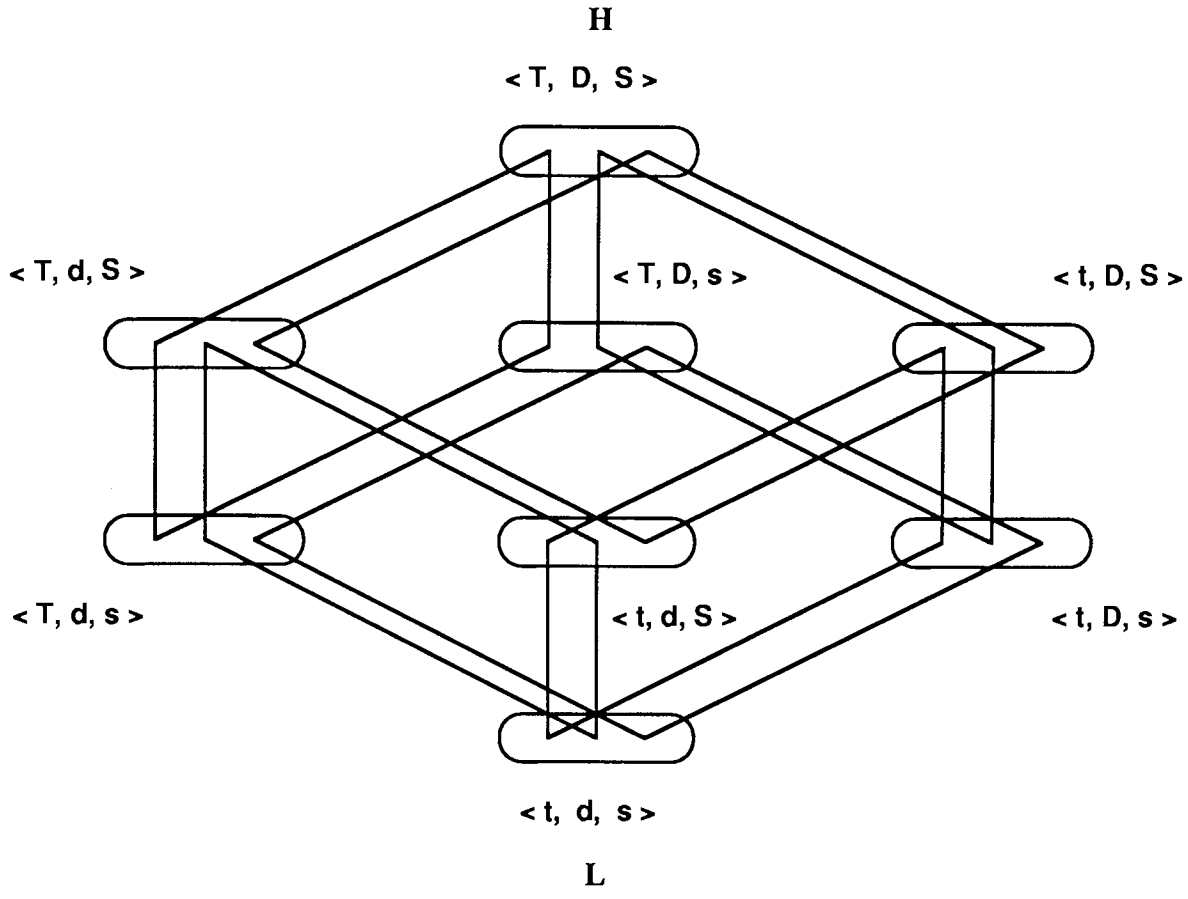


Figure 3.31 TDS-Triples complexity relationships.

Hence, if we look at the three triples which are top-down (upper row in Table 3.8), but not specifically high level, then it was not previously apparent that they would not all be "sub-high level!" Similarly, if we refer to the triples which are bottom-up but not specifically low level as "sup-low level", then again, one of them is sub-high level. We take up this work again in the next chapter in more practical ways.

3.5.8 The scope and applicability of learning.

This brings us to the *second major point* (section 3.5.7). The hypercube defines a system for learning. A major concern in systems research is to define the scope and boundaries of the system's application. (We refer here to learning rather than the mathematical scope considered in sections 3.4.6 to 3.4.6.5). For example, one might usefully consider the scope and boundaries of the education system or the transport system.

We have made out a theoretical case for the useful application of the hypercube to the simplest possible learning system, namely, low level learning in the restricted sense we have now defined it by. The question arises, can the hypercube be usefully applied to the higher levels of learning in Figure 3.31. Indeed it can, but if and only if, extreme care is taken (section 3.5.6). We have successfully applied the hypercube to simple examples of such

higher level learning. But this is unsatisfactory. An isolated experiment that just happens to work does not prove the general case. Theory or analysis is required to indicate scope.

Before considering the general case for higher level learning, we must first **lay the ghost in the machine.** The necessity for a priori domain specific knowledge still has some life left in it. Before we start, we introduce a severe restriction. The hypercube is, in principle, a very simple data structure. Without at least a multiple hypercube architecture, a learning algorithm based on the lines already suggested would be incapable of many of the more complex learning tasks inherent in the higher levels. For example, if learning requires a serial reasoning processor then it would be difficult, if not impossible, with just a single hypercube architecture. This restriction restricts application to very simple examples of the states in Figure 3.31.

3.5.8.1 Martin's law and the enhanced Martin's law.

The apparent need for domain specific knowledge is well known to all of us. The reader for example is using a considerable amount of knowledge of both AI and English in order to learn what this thesis is all about. Martin, after a series of psychological experiments, proposed an enigmatic law, (Winston, 1984). Martin's law states that: *in order to know anything, you must nearly know it already.*

But this is highly unsatisfactory. Why does Martin use the word "nearly"? What does "nearly" imply in practise: 90%, 95%, 99% ...? What kind of a law is it that uses the vague word "nearly"? One is tempted to suggest that Martin *nearly* got it right, and that the real answer in an enhanced Martin's law is "completely." But that would bring the recursive devil of assumed domain specific knowledge fully back to life! Not so. We argue that quite the reverse is the case. The paradox is escaped by crucially considering what we mean by the word "know."

We recall Minsky's advise that learning is a non-problem (section 1.1.3). The real problem is representation. Nevertheless, a mere data structure, even one as complex as the higher dimensional space of a hypercube, is *necessary but not sufficient* for a learning engine. Before we leave this point, it is to be observed that the hypercube is a data structure with a unique advantage, namely, that within its closed world context of application, every possible combination exists within the hypercube. The ***universe*** not merely the exemplars ***pre-exists already,*** as does the ***domain independent learning algorithm.*** That is the essence of the hypercube. A hypercube being set up solely from the dimensionality of the problem at hand has the advantage that **the hypercube does "know" it completely, already!**

3.5.8.2 The dangers of anthropomorphism.

But what does it know? Here is the catch. The answer is next to nothing! One extremely simple application of the hypercube which has been tried was in the domain of production tools for Intaglio prints. It worked - even so far as to successfully suggest possibilities not given by the expert. Nevertheless, the hypercube "knows" nothing about tools, Intaglio production or printing. How could it know? Remember, Tabula Rasa's do not work, (section 3.5.1).

What the machine has learnt is at the simplest possible level by a domain independent combinatorial search algorithm. In this sense everything that it eventually "knows" completely pre-exists, but it is not specifically pre-programmed as such. "Learning" in this sense occurs at such a simple level that our use of the word is polysemous with its everyday use, and unless we are awake to this error, is wistfully anthropomorphic.

It is an all too common error in AI to assume more of the machine than is the case. We are so used to implicitly assuming in our conversations with one another that others also know and understand the world and our conversations, that when faced with a computer printout that gets it "right" we unconsciously assume unfounded abilities of the machine. Indeed, in the conventional case, the machine will merely by preprogrammed causation, have come up with a combination which, with the aid of ourselves doing probably far greater than 99% of the real processing required, seductively causes us to pronounce it as "correct!" This is amazing.

In the example of the Intaglio prints the machine has not learnt and does not know what a tool is. It does not even "know" what a word is. It does not itself "know" what a symbol is - for example the ASCII code it uses. It works mechanistically shuffling binary. But it is even unaware of this level. It does not "know" that it uses binary: the machine simulates binary by a "high" and a "low" of which it is also unaware and which the hardware below its software horizon simulates by the five volt and zero volts (actually 0.4 to 1 volt and 4.3 to 5.5 volts in most modern workstations) which is at the lower hardware levels. Small and medium scale integration automatically switch the logic circuitry. Nor does the machine "know" this. The machine is "aware of" and "knows" practically *nothing* in our sense of the word, hence we advisedly use the word anthropomorphic to describe our delusions. All the rest is our imagination!

3.5.8.3 A machine interpretation of learning.

Yet the machine learnt, by example, what we interpret to be the correct answer for the Intaglio prints. It did so because a necessary but not sufficient reason was that the

hypercube was set up to completely know it already! Defining the total scope of the problem from the dimensionality of the problem, already allows for all possible combinations. Nevertheless we should be very cautious, as shown above, not to read too much into this. The "magic" is mainly self-generated and self-deception and therein is the source of the paradox of our enhanced Martin's Law.

Assumed domain specific knowledge is not necessary. Parsimony suggests that given the dimensionality of the problem, a domain independent algorithm inductively generating the simplest possible globally biased generalisation and some known exemplars will result in sufficiency. Why should we bother with the unnecessary? Let us reiterate the point. If representation, for a given hypercube dimensionality, is necessary but not sufficient for learning, bias together with domain independence provides sufficiency, given some known exemplars, in the form of the actual mechanism for prediction and hence provides a *process* based upon the subhypercubing algorithm by which *we interpret* the machine to have "learnt" from the particular examples.

That is, once we have defined the relevant hypercube, the machine implicitly completely knows it, already! Only the storage of the particular known examples and their biased interpretation, as above, remain. After which process we use the words "learn" and "knows" (Strictly in a more restricted and polysemous sense to our human use of the words). This is often the case with technology, whereby a word is used if the approximate effect is achieved, rather than it being used for the same effect only. For example, we say that both birds and aircraft "fly." But the word and underlying principles of flight are different in the two cases. Bird flight does not typically rely upon the fixed wing aerofoil uplift principle of flight used by most aircraft.

We conclude that the hypercube both can and cannot learn! It *can* learn because we have redefined the meaning of the word learn! This meaning is now precise. It *cannot* learn in our normal sense of the word. [16] Our normal meaning of the word "learn" is very fuzzy from a computational point of view. Since we don't know what the word means with any precision we are unable to build such a machine. We repeat, it is often inadequacies of language that is the source of our problems (section 3.5). So, finally we have answered our question (section 3.5) "what is learning?" with respect to the hypercube.

3.6 CONCLUSION.

Section 3.1.3 outlined the basic requirements for this chapter which developed a theory of hypercube learning (for further details see Ball & Harget, 1989). We initially required solid links into experiments on reality concerning some fundamental, eternal fact

16 For example, learning also often seems to include understanding - whatever that might be!

about learning. That was the psychological experiments. We then developed a complexity range theory of the hypercube and a more powerful version, the subhypercubing algorithm. Enhancements were then made to this algorithm in the form of eleven requirements. These behavioural advantages of the basic algorithm formed the theoretical predictions of the theory. We also considered the theoretical scope of what the machine is capable of representing mathematically and limitations of the machine with respect to learning. The theoretical predictions ought to be tested, so we will next consider practical matters.

CHAPTER 4.

The Subhypercube Machine: Development and Experimental Results.

4.1 INTRODUCTION.

Research projects in the field of learning typically involve the application of some accepted theory such as KOHONEN NETS, ID3, WISARD, BP, etc. This accepted theory is usually applied in some new application area. The basic theory of the machine is assumed. Occasionally the project concerns some extension to the theory. Again, this work can build upon the existing basic theory. Very rarely, some whole new theory is conceived. We hope that it is perfectly clear to the reader that this project concerns just such a novel theory. In contrast to the above situations, our theory has had to be developed "from scratch". Several unfortunately necessary consequences ensue. [1]

Firstly, we have a major task in developing a practical implementation of the subhypercube theory. When building upon some accepted theory and practice it is unnecessary to develop "from scratch" a workable version of the theory. Others have done this already. Also the basics and details of the practical algorithm are known and known to work. It is a considerable step to arrive at this position. Many *cul de sacs* have to be eliminated. Our set task requires the development of as many versions of the algorithm as it takes to produce a debugged, reliable, fast, well honed machine with wide applicability. That is, if the subhypercube theory is to be equipped to achieve similar standing to accepted theory. We found it necessary to overcome many practical problems *en route,* for example: false starts, mistaken ideas, tight loops, beautiful but unworkable ideas, elimination of tempting ad hoc solutions, the fight for parsimony and orthogonality, ensuring that the algorithm really does reflect the theory, etc. In short, in comparison with more typical projects, the totality of the above necessitated that the "**development of the subhypercube machine**" was a major exercise. It is also required that this developmental work is in done accordance with the guide-lines of our scientific cycle outlined in point 3 of section 3.1.3.

Secondly, the purpose of experiment lies in "**testing the predictions of the theory**" (section 3.1). This we do in accordance with our scientific cycle outlined in section 3.1.3. Yet experiments cannot, of course, prove the theory (section 3.1.2). We merely, at

[1] It is "unfortunate" for the serious researcher, since as shown below the penalty to be paid is a considerable increase on the expected standard and volume of work in order to justify comparison with established systems. For this reason alone it is neither a recommended nor a typical approach taken in research work. Further, unnecessary proliferation of theories is also to be very much frowned upon in the absence of a "watertight" first cause. (Chapters 1 and 2 were our justification).

best, retain the theory pending more exacting experimental demands. Again, for a project building upon existing theory this step has already been done by other people. Workable versions of the machine already exist in accordance with the predictions of the theory (but this last step may be poorly done if the theoretical predictions are inexplicit).

A third consequence for the subhypercube theory is that we must "fight for its survival". In building upon accepted theory the researcher can liberally quote the equations developed and checked by others and their practical validity and applicability is not immediately suspect. For the subhypercube theory we have a second major exercise demonstrating the practical applicability of the theory. We must therefore address the **"scope and limitations of the machine."**

Fourthly, for accepted theory, both the theory and its corresponding practical implementation have typically been reworked in the research of many others. For example, we estimate that ID3 and its variations has featured in the work of dozens of people. Back propagation has been used as the basis of hundreds of research projects. Any *new* theory is therefore up against the "sailing boat effect," whereby the new has to catch up with existing technological refinements. [2]

It is, therefore, very difficult indeed within the scope of *one* project for any novel theory to immediately out-perform existing technology or even find favourable comparison. We consider it a major achievement of the subhypercube theory that in its practical implementation, to be described later in this chapter, the subhypercubing algorithm has been found able to *easily* out-perform two recent fast versions of back propagation, even in the worst case of parity and especially for higher dimensional (harder) problems. In summary, we are fourthly duty bound to make explicit the **"specific advantages of the machine."**

Fifthly, the future of any novel machine is still not assured even if it has been demonstrated to: implement the theory in the "development of the machine"; operate in practice according to the "predictions of the theory"; identify the potential for usefully wide application areas by a study of the "scope and limitations of the machine"; show that there are "specific advantages of the machine" over existing technology. All this merely arrives at the ***starting point*** of the practical work of most projects based upon existing technology. A comparison with more conventional research work is still required. *Such is the price paid for challenging the status quo!*

Typical projects may demonstrate some impressively advanced application of the machine in the "real world". All too often, in retrospect, this demonstration appears "overhyped" suggesting greater attributes of the machine than is later found to be justified. A

---

2 Steam boats, although potentially a superior technology, took several decades of development before they were able to oust the existing and highly developed technology of sailing boats, hence the phrase.

very common entrance to this trap may be due to the *lack of a rigorous scientific approach,* as follows. The target problem and the algorithmic solution are allowed to "grow up together". The result is a heavy coupling of the solution with the particular problem addressed. The solution has become by clever programming, tuning of ad hoc fiddle factors etc. highly attuned to the problem. The demonstrated solution does indeed "work like magic" on the problem. It therefore comes as a great surprise to find that on even slightly different "real world" data the "solution" almost invariably fails. The "solution" is often found to only work reliably on "toy world" problems, despite the attempt to target it otherwise.

An example of this effect is the "Blocks World" work of Winograd as later magnanimously admitted by Winograd (Winograd and Flores, 1986). Where does this scenario go wrong? And why? Perhaps we ought to leave this as an exercise for the reader? More seriously, we feel that the answer to this crucially important problem cradles the whole future credibility of AI and in many cases clearly lies in:

1. the lack of a rigorously enforced scientific cycle (section 3.1.3) as given away by the phrase "growing up together,"
2. or caused by unrecognised "hidden variables" (section 3.1),
3. or difficulties are underestimated by many orders of magnitude.

Our eyes have been opened. To produce **"an advanced application of the machine"** which really is "real" is very difficult. The above five consequences together with a "conclusion" map out the remainder of this chapter. After some "preliminary considerations" we therefore discuss each of the following above highlighted items in turn.

1) Development of the subhypercube machine.
2) Testing the predictions of the theory.
3) Scope and limitations of the machine.
4) Specific advantages of the machine.
5) An advanced application of the machine.

4.1.1 Preliminary considerations.

In order to set the scene for later work and thereby not disturb the run of thought in later sections with side issues we gather together under this heading of "preliminary considerations" all such discussion. This includes the hardware used, the language used for the project and its software environment and the reasons for these choices. We also discuss the accuracy and reliability of results taken in the course of development and experiment and the reasons for the choice of the formalism behind the resulting algorithm as against alternative possibilities. We will now address these side issues before entering the proper discussion of the practical work undertaken.

4.1.1.1 Hardware considerations.

The details of the hardware used for this project are discussed below. When the project began the SUN workstations were:

a) designated for research purposes only,
b) the fastest machines to which access was freely available,
c) provided with a good range of software, either bundled or available as packages.

There was no pressing reason for choosing any other facility. It became necessary, in order to keep up with technological changes, for the department to update these machines both in terms of operating system software and with newer releases on the hardware side, although always with continuity within the SUN workstation range. The above three criteria still hold for the present SUN machines within the department, thereby justifying the initial choice. (In the meantime due to the evident excellence and quality of the machines SUN has become the world's largest producer of workstations).

In the course of the project, departmental updating of the computer facilities necessitated the utilisation of four SUN workstations: namely, "Pollux," "Castor," "Rigel" and the "SPARCstation 1". [3] The department has 16 SPARCstations of which four are nominally designated as research machines. These four machines further each act as a server to three of the remaining 12 SPARCstations. Hard disc space on these machines is reserved purely for system software, all user hard disc memory is networked onto the primary university dedicated fileserver called UHURA.

UHURA is a parallel processing machine networked to secondary fileservers such as Castor by the "yellow pages" service. Castor itself, via an Ethernet, oversees the 4 SPARCstation servers, as above, in the event of UHURA malfunction. The SPARCstations are of RISC chip design whereas the earlier Castor and Pollux were 68000 based processor design. Further details of these machines are contained in Table 4.1.

name	model	speed	processor
Pollux	3/50	1.5 mips	68020
Castor	3/160	2.0 mips	68020
Rigel	3/60	3.0 mips	+ fast clock
SPARC	4/60	12.5 mips	RISC

Table 4.1 Details of the research project hardware machines used.

3 Pollux and Castor were departmentally named after the "heavenly twins" in the Milky Way.

4.1.1.2 Software considerations.

The choice of software for the project was far more problematic. Our major concerns were as follows. The machine had to be able to automatically and reliably construct an *arbitrarily* large lattice of nodes and arcs in the form of a hypercube. It was assumed that a block structured language entailing facilities such as the use of pointers would be likely to produce horrendous debugging problems. Failure at this point, would be catastrophic for the project. This was felt to be the major consideration.

In retrospect this point is not so important, but this is only apparent after having built the machine. Having found out how to construct such representations it is perhaps now possible to consider recoding the algorithm in a block structured language without apprehension.

The second consideration was the expected complexity of the algorithm. The subhypercubing algorithm is theoretically **deterministic** for finite hypercubes with defined start and end points in its convergence and cannot, therefore, "get lost" or enter an infinite loop. Yet there is a further problem. In section 4.1 we discussed the necessity of a thorough "development of the machine" stage for all novel machines. Due to the expected complexity of this crucial stage, we wished to avoid all unnecessary distractions from the developmental task at hand. In order to be able to concentrate on the algorithmic problems without the distraction of language implementation details (at least in the first instance), it was felt necessary to use as high a level language as possible.

Prolog has the advantage that it separates the "what" from the "how". Prolog's declarative style ensures that the user is able to concentrate on "what" he is trying to do. Prolog's logic then automatically takes care of the "how" details. This facilitates concentration on specifying the problem. Prolog then does all the rest. This was just the sort of language we required. In a typical high-level language, encoding the "how" would ensure that the program would extend in size (we estimate from some experiments) by about four times.

Given the complexity and length of the resulting algorithm *we made a very sensible choice* to steer clear of conventional languages for this crucial developmental phase, bearing in mind its expected and very fully justified resulting complexity. Prolog is especially well suited for symbolic rather than numeric computation and for solving problems involving objects and relations between objects (Bratko, page 3, 1986). This could almost be a definition of the hypercube. Our choice of Prolog was fortuitous to the success of the project. Our preference was for the best Prolog which is QUINTUS PROLOG. A number of other research students were also interested in QUINTUS and the department duly obtained it.

Nevertheless, there were two further very high-level languages available to us namely LISP and Pop 11. LISP has very attractive features but the lack of software back-up in depth at the start of the project mitigated against it. Bare LISP is not much use, it is all the "add-ons" of, for example, Common LISP (which we now have) which really count. Pop 11 was the initial choice but we quickly became disillusioned by its weak software engineering aspects. Pop 11 contains too many ad hoc unstructured extensions of the enthusiastic "fiddler".

A third consideration was speed. The subhypercubing algorithm is basically an exercise in handling complexity in the fastest possible way, in order to stave off the overwhelming expense of the resulting exhaustive search. This would strongly imply a language such as C, for greatest speed, or Fortran, for its specialised scientific problem speed and wealth of scientific library back-up. For reasons we consider later, these libraries may not be much help. Further, such languages clash with the requirement for a very high-level language. Which is the more important consideration?

We decided that, in the first instance, the job of research is to demonstrate feasibility, not to implement a fast near-commercial system. Hence we decided to ignore this enticing requirement. Simple comparisons between QUINTUS and C suggest a possible speed-up by recoding of up to 10^3 times. This could turn the work into a valuable fast product and is to be recommended as a project for further work.

Considering the software environment, our present version of the hypercube system runs on the SPARCstations and is written in QUINTUS PROLOG which is invoked in the SUNVIEW window system which itself runs under SUN's latest version of their UNIX operating system.

4.1.1.3 Accuracy of the runtime statistics clock.

After the theory had been encoded in a programming system, a large number of experiments were performed for various purposes. Many problems were initially found in respect of the accuracy and repeatability of the readings taken. Since this is a side issue we will deal with it first. For Pollux and Castor departmental changes involved considerable evolution of these machines. The main stages of which we will briefly discuss.

At stage 1: Castor was a dedicated SUN 3/360 research machine.

At stage 2: Pollux was bought as a discless 3/350 node. Pollux and Castor were soon networked for the secondary and later primary purpose of acting as filestations for several rooms of other heavily used PC machines such as Apple Macintosh.

At stage 3: Castor, became the secondary fileserver for the whole Campus using the SUN "yellow pages" (as in section 4.1.1.1).

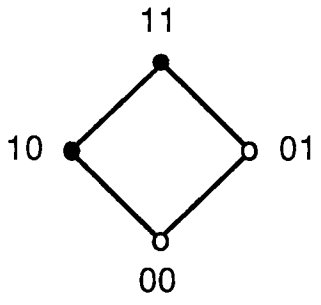
At stage 4: The department obtained a faster machine (it had an on board 68000 based maths coprocessor) which was named "Rigel".

At stage 5: the department obtained 16 SPARCstation 1 machines which were then networked together (as in section 4.1.1.1).

This evolution resulted in considerable perturbation of our application's time slicing due to prioritisation and the relative levels within UNIX of the software in question (the runtime statistics clock in QUINTUS PROLOG). In the worst case this resulted in the same experiment producing results differing by several orders of magnitude! Since much of our work concerned the *accurate* comparison of the runtime of different algorithms on the same problem, this meant considerable disturbance to the work.

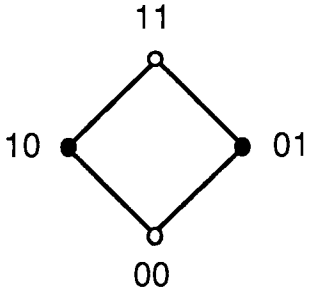
With the advent of Rigel and particularly the SPARCstation 1, this effect has been reduced to very acceptable levels (barely detectable). We found the most reliable clock to be the "runtime statistics clock" within the QUINTUS PROLOG. Further, the QUINTUS PROLOG itself, has interpretive and compilation overheads which also had to be considered.

This variability necessitated a minimum of five or six readings for all the remaining work reported herein. These six readings were needed in order to obtain a more reliable *single* reading with its associated error, as used in all later experiments undertaken. Eight sets of 30 readings each were taken for the four machine types: Pollux, Castor, Rigel and the SPARCstation 1. These experiments were each of two types: a) defining a line in H^2 and b) parity in H^2 as in Figure 4.1.



Index	Class
11	1
10	1
01	0
00	0

(a)



Index	Class
11	0
10	1
01	1
00	0

(b)

Figure 4.1 H^2 runtime statistics experiments.

Similar experiments were repeated for H^3 and H^4. We were interested in finding out if there was any spread, and if so whether it varied with the dimensionality, the problem difficulty (in this case a line as against parity), what the QUINTUS PROLOG overhead

might be and what effect different machines might have under different external loading conditions, etc. We also experimented with the complementary parity. The approach taken here and in later experiments was on observing an effect to test its reproducibility in further experiments.

The QUINTUS PROLOG overhead was found to be measurable even for further experiments involving the simplest possible one line program. The overhead was greatest in the case of Pollux and least in the case of the SPARCstations, as expected. By the term "overhead" we mean that a certain minimum time is required for any such computation. The overhead for Pollux, for example was found to be 170 msec greater than Castor (possibly a measure of the combination of disc access seek time, network access time for the discless node and the slower processor time). All readings for all experiments were always found to **quantize** into distinct groups forming a symmetrical graph. The system clock quantization was found to be unerringly 17 ± 1 msec for all four machines. Typical graphs obtained are given in Figures 4.2 for Figure 4.1 (a) on Castor (± 17 msec bars); and Figure 4.3 for Figure 4.1 (b) on Pollux (which looks much more like the normal distribution that one might expect).

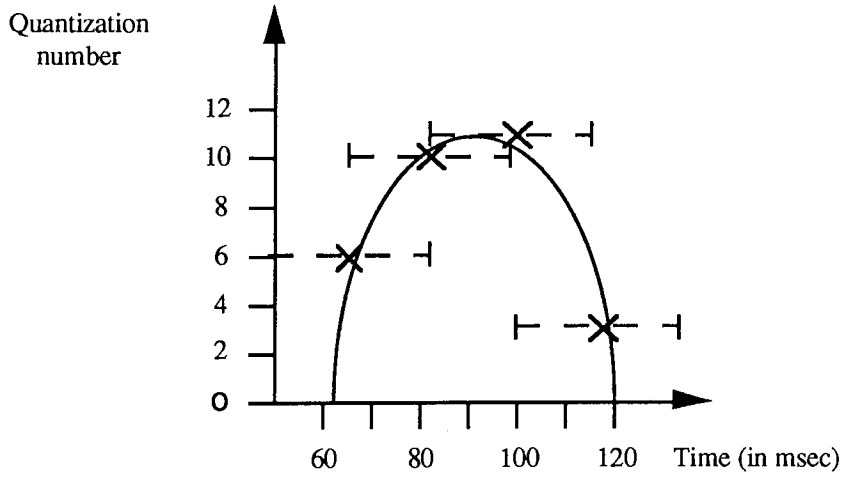


Figure 4.2 Castor quantization.

The spread for Pollux was much greater than for Castor, Rigel or the SPARCstations, which were similar. Further, for Castor and Pollux only, the spread got worse with increasing dimensionality. This problem was by far the worst on Pollux (approximately proportional to the total time). Pollux was found to be so annoyingly variable and in addition had the severest fileserving overhead (referred to above) that its use was discarded. Castor also proved to be too variable for all serious research. Nevertheless, against this background all initial work on the development of the algorithm took place on these two machines. This necessitated a very large number of repeat experiments (in excess of the multiplicative six readings referred to above) during the most quiescent periods, in order to obtain reproducible results.

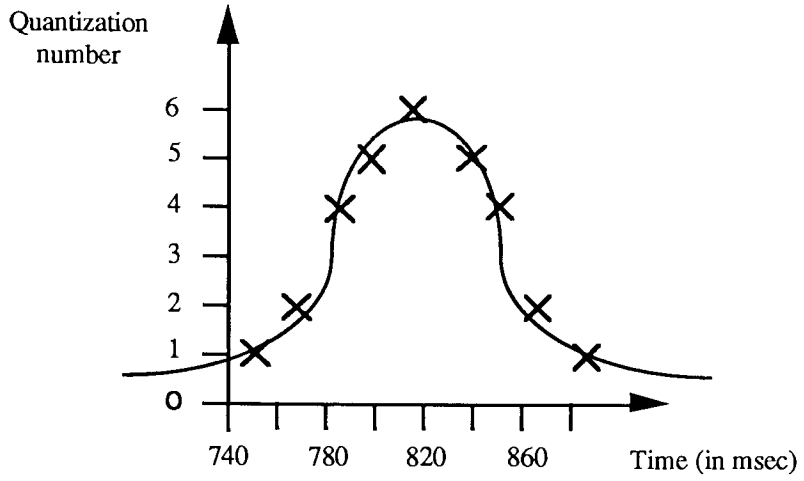


Figure 4.3 Pollux quantization.

Briefly, the conclusion for the final machine environment, the SPARCstations, was that for runtimes in excess of one minute the spread increases little with either dimensionality or problem difficulty. Further, for times less than one minute, for all practical purposes we may assume that a clock quantization of two clocks that is 2*17 msec = 34 msec is sufficient to cover ± 2 standard deviations. We are justified in quoting runtime, X, well in excess of 1 minute as X ± 60 secs. [4] For the reasons given it is difficult to give a comparable error factor in the case of Castor and a waste of time for Pollux.

Rigel and the SPARCstations were similar, the later being dependably a fraction faster for large problems, due to memory management difficulties on Rigel. These problems were, however, given enough experiments, partially ignorable in practice, since our main concern was to relatively compare the runtimes of different algorithms in the development of the machine. Where these differences were large, particularly in the earlier stage of algorithmic improvement, the ascendancy of one algorithm over another was clear enough even allowing for the annoying lack of a precise measure of the error involved. Where this was not so, life became more difficult. (Possibly Pollux could serve renewed purpose in the physics department as an excellent working demonstration of Heisenburg's Uncertainty Principle).

Lastly, parity was found to form a "double hump" graph as Figure 4.4 for Castor, where one peak corresponds to odd parity, the other to even parity overlapping in spread for the simpler problems. This was expected. Although the theory regards both parity functions to be of the same type, in the practical implementation of the algorithm, they are bound to be distinct but close in time, due to practical considerations such as the order in which the

[4] Nevertheless, a case could be made for taking the minimum rather than the average. In which case all results quoted below in the rest of this thesis would improve somewhat. The argument is that the minimum time identifies how long the run takes under ideal conditions; the fastest time can be regarded as being a true measure of problem complexity in the absence of the disturbing factors of extraneous system loading.

algorithm searches through the subhypercubes, algorithmic details etc. For example, the "top bot" (see later) technique which exponentially speeds up the algorithm has dimensionally problem dependent subhyperface sizes, thereby impacting upon the parity type.

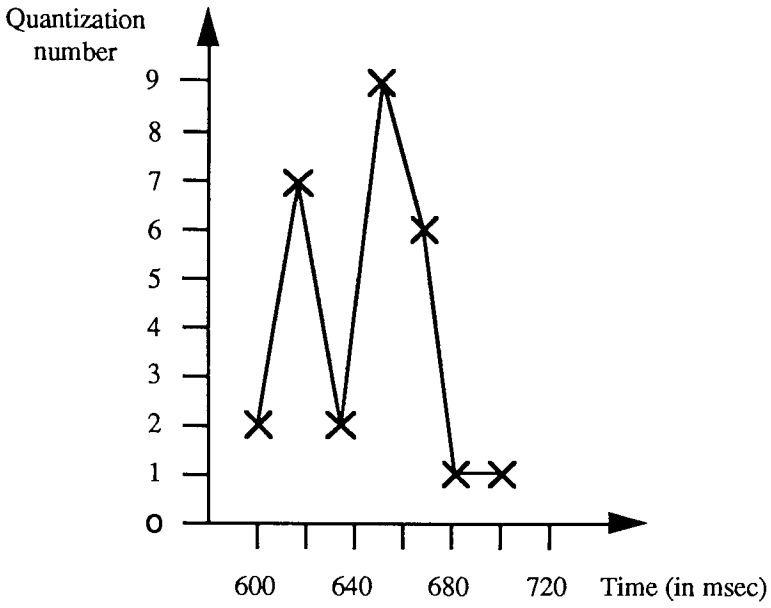


Figure 4.4 The parity "double hump."

4.1.1.4 Karnaugh mapping.

In order to report what it is that we have found from experience to be a great weakness with the Karnaugh mapping approach for our present purposes we will present the heart of the problem in the abstract. Given a two dimensional surface let us subdivide it, as Figure 4.5, with subareas numbered 1 to 9.

1	2	3
4	5	6
7	8	9

Figure 4.5 Karnaugh map.

Next let us represent Figure 4.5 as a one dimensional line with the subareas as before see Figure 4.6 (a). If we consider an adjacent grouping in Figure 4.6 such as 4,5,6 then Figure 4.6 (a) similarly groups 4,5,6 adjacently; but if we consider the vertical adjacent grouping 2,5,8 in Figure 4.5 then Figure 4.6 does not represent the information as an adjacent group. In a "**clustering approach**" to learning it is necessary to find some basis by which we might search for the discrete parts of the cluster or group. This is a well known

problem. A simple example of the effect within the same dimension is the difficulty of accessing information by row if the information has previously been organised by column.

A partial solution (which works for small problems) is to use **Gray coding** rather than binary, since Gray coding maps **Euclidean neighbourhoods** into **Hamming neighbourhoods** (one bit apart). For the first few dimensions there appears to be little problem. But as the dimensionality rises beyond the tenth dimension then it reveals itself as exponentially ever more cumbersome to "group" by the apparently necessary "**folding complexity algorithm**" that we developed. Further this effect occurs "**multiway**" (in all four directions from any given cell within the mapping) and since the size of the mapping is itself growing exponentially we have a "double exponential" effect. This is not a problem that we have seen reported elsewhere. (Although the general cumbersome nature of Karnaugh mapping, for more general purposes, beyond the eighth dimension is commonly known).

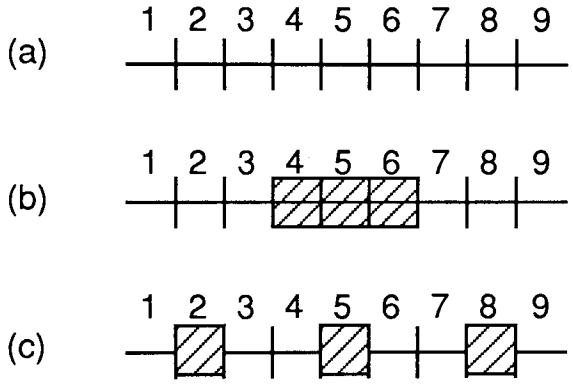


Figure 4.6 Linear map.

Again, representing information in the third dimension in a two dimensional space will similarly lose relational information such that processing is required to rebuild the relational information that is "ready to hand" in the third dimension: even more so for 3rd -> 1st dimensional representations. Clearly, in the general case of some high dimension D -> 2nd dimension huge amounts of information are lost or require enormous computational efforts to recapture it all. *If this were not so there would be no advantage in higher dimensions!* But the world does have higher dimensions, possibly as high as the tenth (section 3.2.4) and we pay a high price for our representational paucity.

On the other hand if the information is represented dimensionally by a representation of the *same* dimensionality then we eliminate the need for the above computation, since however fast this addressing computation is performed it is still in excess to any other computations performed later. This effectively prohibits consideration of very many standard approaches to similar work such as linear programming; e.g. the use of matrix algebra

together with Fortran and its associated libraries such as the NAG library. (We are, of course, concerned with *non-linear* programming - it being the essence of AI - herein). [5]

It seems that we are forced to make use of the full dimensional representation of the hypercube and develop an appropriate algorithm. (Thus 'the penny dropped' and, as we saw in chapter 3, we did just that). There are, however, two main possible directions (section 2.1.2.5, section 2.2.3.3) in which the hypercube can then be searched: 1) bottom-up or specific to general, and 2) top-down or general to specific. We consider first our specific to general research work.

4.1.1.5 The specific to general: getagroup.

Getagroup was our first implementation of the cr theory in section 3.4.4.1 resulting in the most complex algorithm we have ever attempted. Its intricacies are conceptually horrific. The algorithm remains incomplete and is now discarded. It was inspired partly by the possibly mistaken belief that actual neurons may employ a similar bottom-up approach. Bottom-up design of the cr theory has certain attractions with respect to efficiency and speed. As Ratcliff (1987) states, on pages 74 to 75, when discussing various approaches to software engineering design and bottom-up design in particular, bottom-up design is:

> "... especially useful where complex, innovatory system development is involved, bottom-up design may provide the only way 'into' the problem ..."
>
> "... bottom-up design should not be regarded as in any way inferior to top-down design, provided it is appropriately applied."
>
> "It is the mirror-image of top-down design, involving the structuring of progressively higher levels of abstraction. Thus bottom-up design is a process of synthesis rather than decomposition that hides, rather than reveals, detail."

Our bottom-up implementation of the cr theory did indeed provide, as Ratcliff states, a "way 'into' the problem". In this sense the getagroup exercise was an invaluable first step towards an appreciation of the problem. Nevertheless, the disadvantages that Ratcliff cites for bottom-up design such as the difficulty of maintaining correctness, the lack of a global viewpoint until completion and the lack of associated evaluation criteria when there are numerous intermediate levels of abstraction were ultimately telling. As stated we failed to complete the algorithm - although it is very close to completion (the system is "completed" in a sense, that is, it is not completely specified for all possible inputs, certain modules and procedures remaining outstanding). The system is very fast: 3, 5, 10 secs. for parity in H^4, H^5 and H^6 respectively, on Pollux! It is therefore worth, for completeness

---

5 While we are discussing this subject, another point is in order. A most interesting aspect of our research is that if we take this matter further such that the (hypercube) representation contains excess dimensionality then considerable new advantages accrue: intractable non-linearities become tractable linearities (see later).

sake, discussing some of the major components of the algorithm: but in order not to distract from the thread of the discussion this outline of the algorithm is relegated to Appendix 5.

4.1.1.6 Hypercolumn connectionism.

As we remarked in section 4.1.1.6, the getagroup algorithm is an implementation of the cr theory in section 3.4.4.1. The difficulties of the getagroup implementation encouraged us to search for a more useful theory than the powerful cr theory. This search resulted in a top-down perspective which led to the subhypercube theory of section 3.4.5.3. What happened was that the rejection of the bottom-up approach led on to a concept of "**hypercolumn connectionism**" as opposed to "**neural connectionism**". Developing these ideas led to our present concept of the hypercube. Again, rather than disturb the flow of this report, we leave these details of this discussion until Appendix 6.

Thus the hypercube representation is an attempted abstraction of the hypercolumn (section 3.2.6). Attention then shifts to the realisation of the analogous hypercolumn components as the shape of the solution set within the hypercube. The subhypercube theory naturally leads to a convincing abstraction of the hypercolumn components as 'polytopes' or subhypercube groupings (section 3.4.4.1). In contrast, in the cr theory and 'getagroup', if one were to attempt to model the hypercolumn then it would have to be generated bottom-up by its neural equivalents and the hypercolumn components which are opaque. We will now discuss the implementation of the subhypercube theory.

4.2 DEVELOPMENT OF THE SUBHYPERCUBE MACHINE.

The development of the machine went through several stages. We have already discussed the earliest attempts: Karnaugh mapping and the bottom up approach of 'getagroup'. Since these approaches were unsatisfactory, the search direction was changed. The search direction was changed to a search through the subhypercube heterarchy both from the most general and the most specific ends and thereby coalescing upon the solution in between. We will shortly consider the basic algorithm and its refinements in some detail.

4.2.1 The Hypercube system.

Firstly the subhypercube machine exists in an environment which is the complete hypercube system. In the block diagram of the hypercube system in Figure 4.7, the subhypercube algorithm is contained in the block called "learning machine" seen at the lower right hand corner of the figure. Each of these blocks in the diagram is composed of several files which contain the various programs required to provide that particular facility within the system as a whole. These programs also form useful subcircuits. For example, in order to

generate and test all possible combinations within the universes of the first four dimensions the following programs each call the next in line and pass files:

exptgen -> autogenerate -> generateall -> learn -> expand -> checkhh1

All these subsystems provide the necessary software which is ancillary to the central thrust of our research which is concerned with the hypercube. They are therefore of lesser interest. Since space does not permit a detailed description of all these blocks and the various files within each we will restrict our report to the subhypercube algorithm called "learn" within the "learning machine" block.

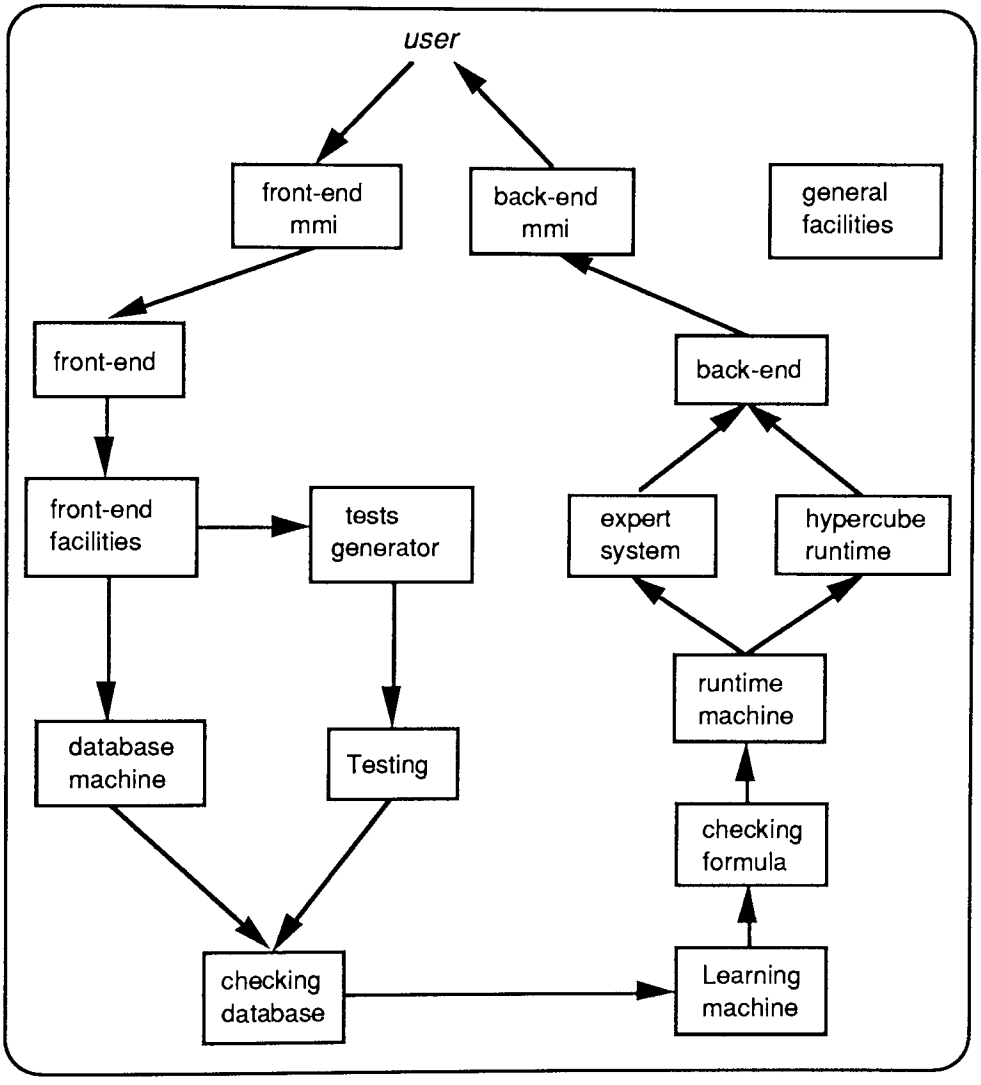


Figure 4.7 Block diagram of the hypercube system.

All software was designed using Jackson structured methods JSD, transcribed into JSP and then translated into prolog. The structure diagrams, however nearly all occupy a complete page. In order to save space and get a better overview of the algorithm we give the generalised structure design on just one page in Figure 4.8; but we have had to miss out all the detail as indicated by "..."; read from top to bottom, left to right.

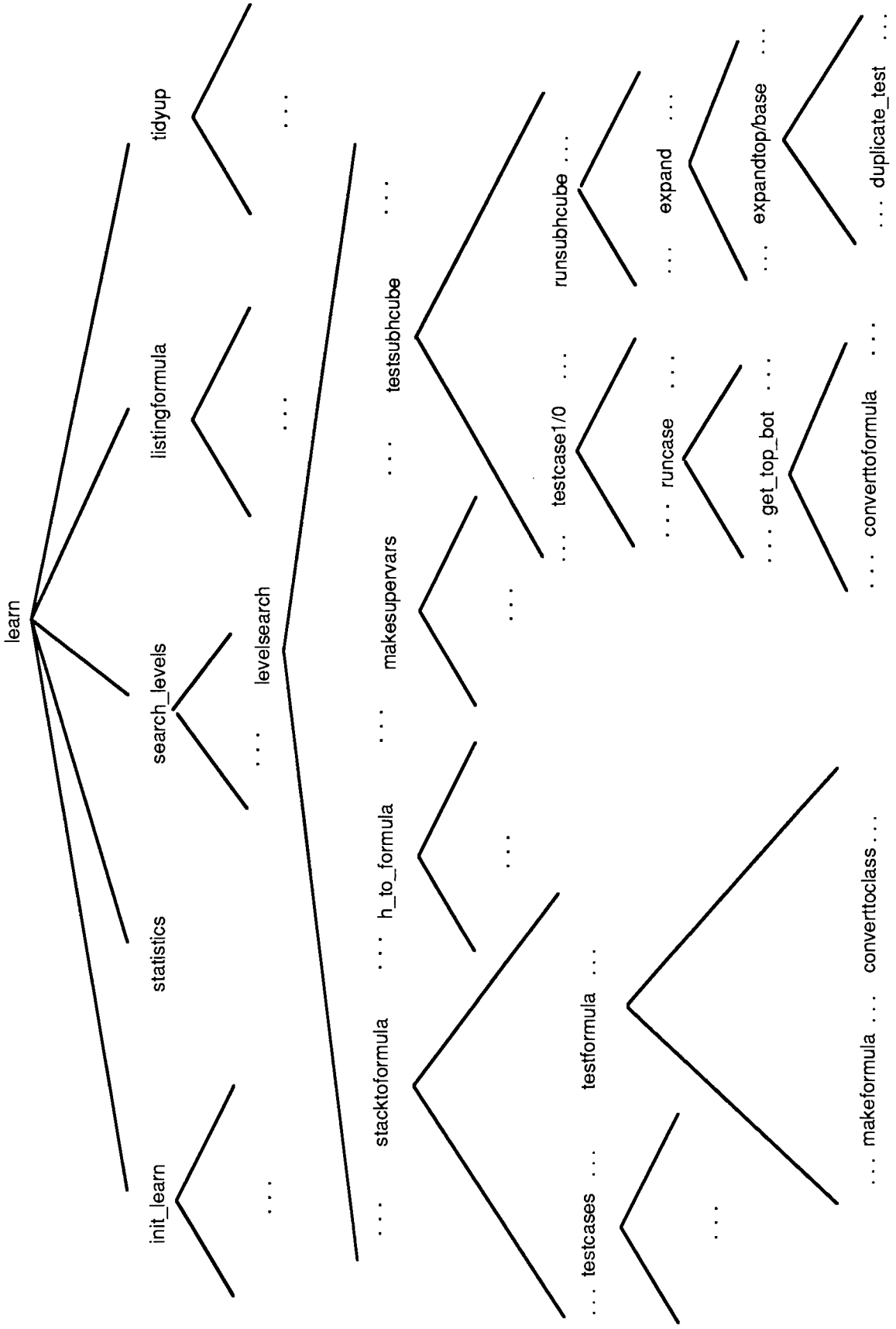


Figure 4.8 Structure Diagram of the "learn" algorithm.

4.2.2 Refining the learning algorithm.

The basic learning algorithm was refined from an initial "**global0**" through **global0.1, global0.2, ... to global0.9.** This final version of the algorithm was later renamed "**Learn**". We now indicate the scope of the approximately 30,000 runs undertaken. For each of these 9 versions, typically about 9 basic types with about 7 subtypes of about 8 Dimensions using about 6 runs/Dimension were performed. (Where "type" means b_chop, Top_bot, parity, etc. and "subtype" means parameter set, trace, etc.). The work described herein is from a representative class mix out of this range. Firstly, we consider a few points about parity. Many problems beside the parity problem were experimented with using the above algorithms, but in our report of the development of the machine (section 4.2) we shall restrict our report mainly to the parity problem. The reason is that we need to have a **standard** by which to compare these algorithms. We chose **parity**. Parity is an excellent and well accepted benchmark for testing connectionist networks. As a defined problem it is: independent of the dimensionality or domain, the hardest possible nonlinear problem within the dimension, it also serves as a datum by which to compare and hone refinements to the algorithm. We have experimented with parity as far as the 15th dimension. We believe that this is in itself easily a record for such systems. The H^{15} parity problem takes the final algorithm nearly a day to learn.

4.2.2.1 The exponential explosion.

It is worth while getting acquainted with the nature of the problem before we start to examine the algorithm and its development. Figure 4.9 illustrates the exponential explosion with the readings scatterplotted somewhat unconventionally as time against dimension. Figure 4.9 was plotted from results taken from runs using global0.7 on the SPARCstation after compiling for the parity problem.

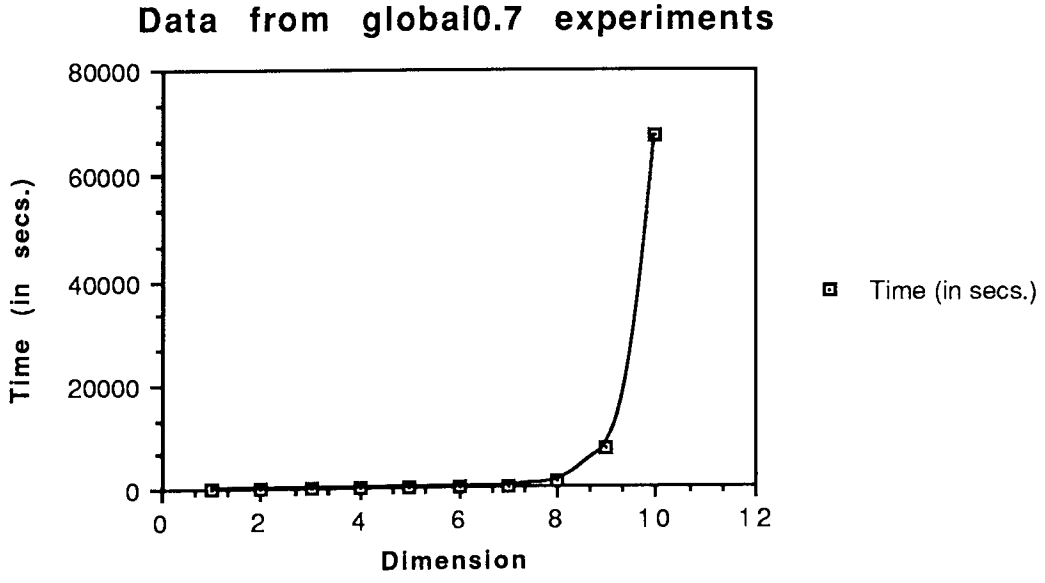


Figure 4.9 The exponential explosion.

One thing which is clear is that the graph "takes off" after a while. The graph at first seems to be well behaved and even linear and then suddenly it runs into trouble. This is an illusion, a snapshot of the lower dimensions on the same data shows the same effect in Figure 4.10. (These and many other graphs that we present are fairly smooth due to the large number of readings taken in order to attain such accuracy. Even so they are not perfectly smooth. There is an inherent spiky perturbation in the graphs quite naturally. This is due to the the fact that the odd and even dimensions actually follow two slightly separate but well-behaved curves).

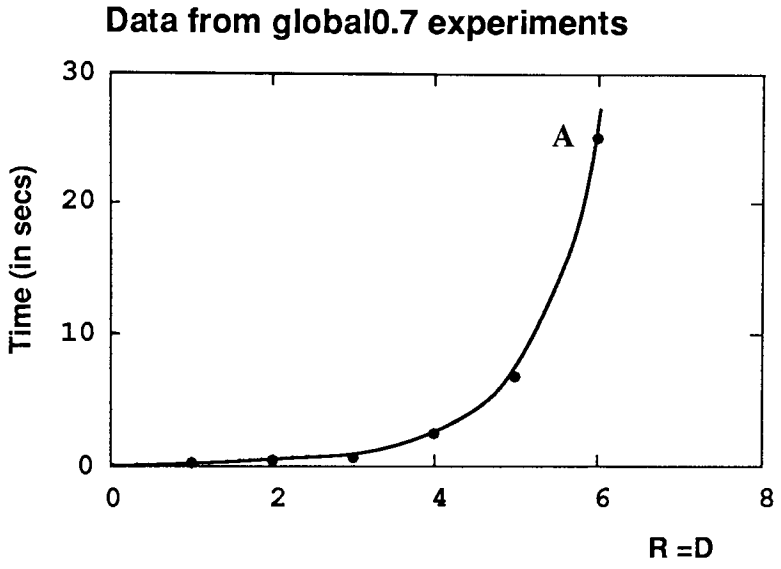


Figure 4.10 The exponential sub-explosion.

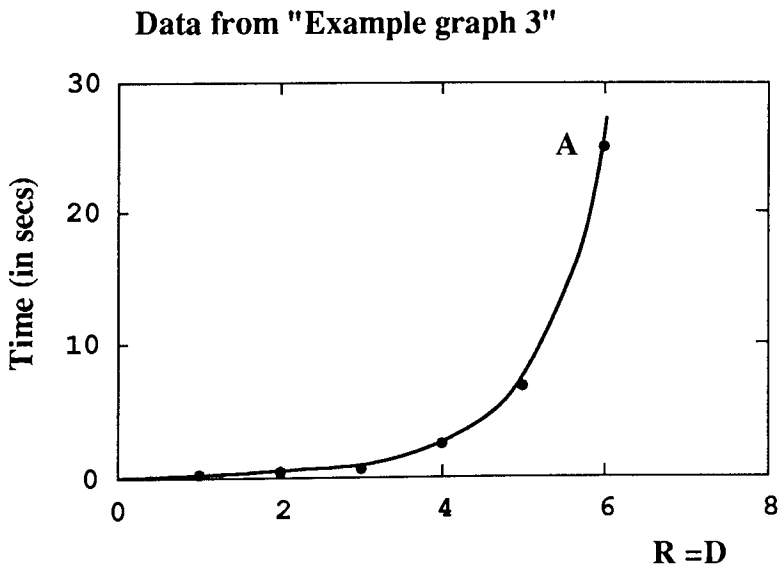


Figure 4.11 Exponential nature of R = D.

The basic question we are concerned with is: "What is the nature of this curve?" In effect: "Which points are the most influential?" A reoccurring process in developing a faster and faster subhypercubing algorithm has been to continually ask this question and find by various means where any inefficiency still exists. Rather than spend many pages discussing the matter in detail with respect to possible refinements and their various effects, let us instead illustrate these "influence points" graphically. Figure 4.11 was plotted from results taken from further runs using global0.7 on the SPARCstation and this time interpreting. It is instructive to consider the set of experimental results for Figure 4.11 with respect to their influence points, see Figure 4.12. Several points are in order:

Firstly, by the term "influence" we mean that the *diameter* (see right hand side of the graph) of the plotting symbol (a circle in this case) represents the *influence* of that point in the correlation coefficient. (The influence is the difference between the correlation computed with a point included and the correlation with the point omitted). It is clear that 'A' = 3 is highly influential compared with 'B' or 'C'.

Secondly, the Pearson correlation coefficient, R, is equal to zero (it always lies between -1 and +1), that is, there is zero strength of any *linear* relationship between the two variables. The Pearson correlation coefficient was calculated by a) standardising each variable, then b) averaging the product of the standardised values across all cases to give the mean product of the standardised scores in appropriate notation as:

$$R = \frac{S_t^1 * S_D^2}{n}$$

Thirdly, Figures 4.11 and 4.12 refer to the same data (despite slightly different sized graphs). So 'A' in Figures 4.11 and 4.12 are the same point. Fourthly, We have not attempted to put an interpolative curve in the graph on Figure 4.12 due to the fact that it would obscure some of the very detail we are trying to point out.

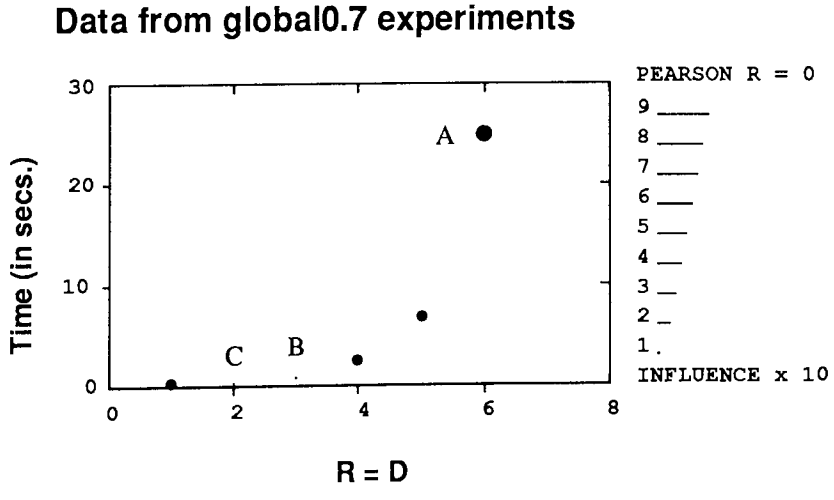


Figure 4.12 Influence points in R = D.

Fifthly, this detail bares close examination. (In fact, *all* experimental results should be subject to close scrutiny, as a matter of good practice, and the maximum information possible extracted from the data). As stated above, we notice how the second and third plotted points on the graph (labelled 'B' and 'C') have almost disappeared One could be forgiven for mistaking them for imperfections in the paper! Whereas the indication is that the final point on the graph (labelled A) as evidenced by its gross size already contains *serious evidence of some further unnecessary explosion due to a slight inefficiency compounding at higher dimensions*. Methods and insights such as this have greatly helped to refine the algorithm during its development stages.

4.2.2.2 Multipliers.

Another way of considering the data is to transform it. If we take the log time ('ln' or log to the base 'e') then the graph in Figure 4.11 is seen to transform into an almost linear graph as seen in Figure 4.13. (Axes are reversed and a greater dimensional range is used).

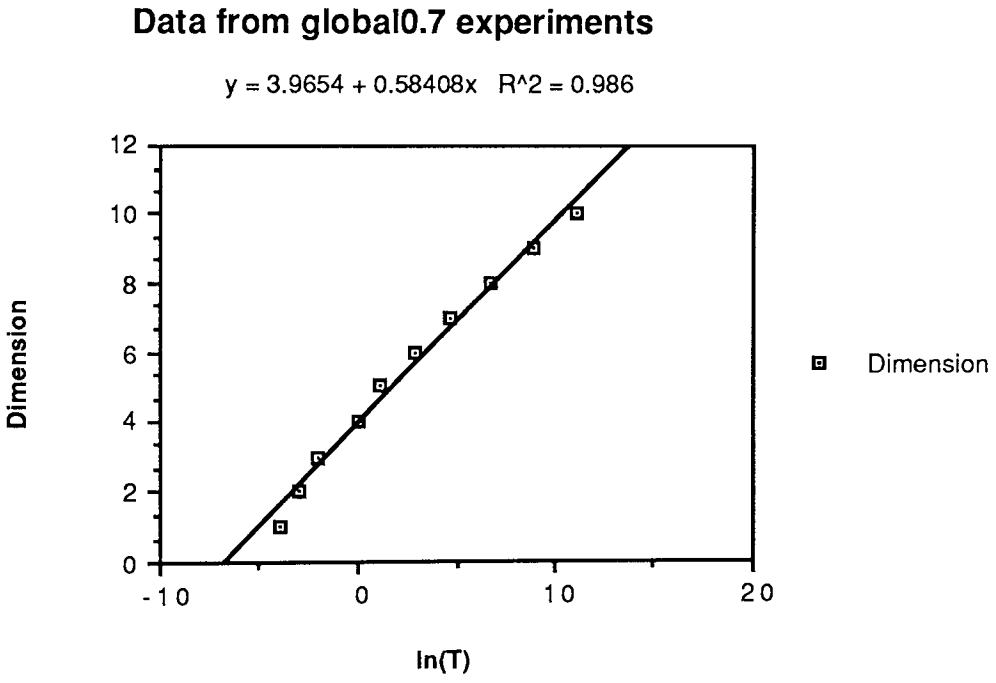


Figure 4.13 Illustrating the logarithmic linearity of the explosion.

This indicates the possibility of some regularity being present. Of course we would hope to find such regularity as predicted by the theory - but we must not experimentally assume it will exist since we are trying to find the extent to which the practical machine embodies the theory. The original data (obtained after interpreting) was a table composed of dimension D and time T (we just quote the average below) in seconds for parity runs of the corresponding dimensions. It is useful now to return to the original data, as given in Table 4.2, and add in some extra columns. Firstly if we take the D = 1 medial value of 0.017

seconds resulting from many runs as a datum and successively multiply the previous value (somewhat arbitrarily) by 4 for each dimension we arrive at the values of the third column. If we now take the log of the second and third columns to create the fourth and fifth columns, these can be plotted against D as shown in Figures 4.13 for ln(T) and 4.14 for both ln(T) and ln(4*T1).

D	T	$4*T_1$	ln(T)	$\ln(4*T_1)$
1	0.017	0.017	-4	-4
2	0.067	0.068	-3	-3
3	0.183	0.272	-1	-2
4	0.85	1.088	0.084	0
5	3	4.352	1.471	1.099
6	16.6	17.508	2.863	2.809
7	102.733	69.632	4.243	4.632
8	822.567	278.528	5.630	6.712
8	7349.433	1114.112	7.016	8.902
10	67373	4456.448	8.402	11.118

Table 4.2 Illustrating the 4T rule.

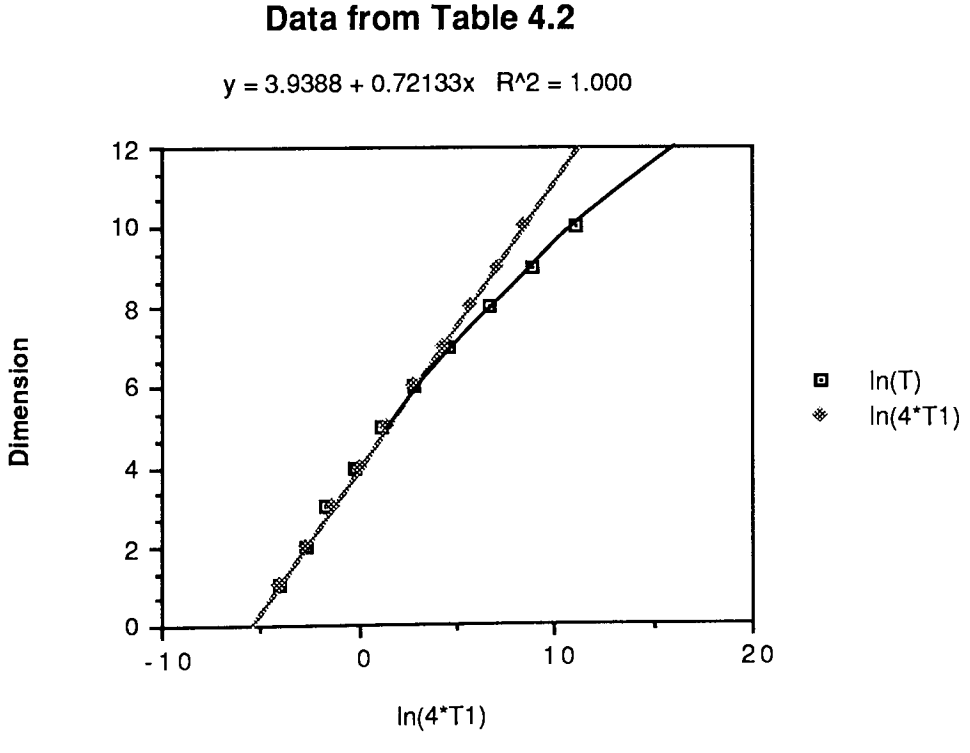


Figure 4.14 The 4T linearity guide.

We now have a most interesting result. The straight line multiple by 4 is also approximately followed by the practical curve for most of the way along the graph - divergence becoming significant with the last three readings. It is a reasonable assumption, born out fully in later experiments that this divergence indicates some inefficiency still existing in the system which needs to be discovered and eliminated. Indeed our "**rule of four**" is well known and has often been postulated as the theoretical limit to work in neural nets. For example, using back-propagation Tesauro & Janssens (1988) state :

> "We find that the training time behaves roughly as 4^n, where n is the number of inputs, for values of n between 2 and 8."

We believe we reasonably compare "like with like" since they worked on the parity function using booleans and the above graphs refer to our work on the parity problem. But are they correct? Is it true that as the number of inputs (and therefore also the dimensionality) is incremented then the time to learn, at best, increases four fold? We disagree with Tesauro & Janssens despite their somewhat vague claim of a "possible theoretical reference" as a backup. Let us see why. If we reconsider the hypercube representation as discussed in the theory chapter then it was stated that the hypercube *doubles* in size for each dimensional increment, *not quadruples* ("**double and join rule**," section 3.2.4). This does not automatically mean that the minimal search path through the subhypercubes will double for the parity function. Remember, our "**double rule**" (section 3.2.4) stated that the effect is a doubling only for *first* level subhypercubes ...

We need to get side-tracked for a moment in order to explain. Our theoretical equation is complicated in the general case. In the general case the maximum number of subhypercubes produced at the Lth level is given by:

$$S_L = \frac{2^L}{L!}\{D(D-1)\ldots(D-L+1)\}$$

and so for the next level we have:

$$S_{L+1} = \frac{2^{L+1}}{(L+1)!}\{D(D-1)\ldots(D+1-L))(D+1-(L+1))\}$$

$$= \frac{2\,S_L(D+1-(L+1))}{L+1}$$

$$= \frac{2\,S_L(D-L)}{L+1}$$

$$\therefore\ S_{L+1} = \frac{2\,k\,S_L}{p}$$

where k is the difference between the dimensionality and the level, and p is the next level. Thus the *subhypercubes* multiplying factor is 2k/p. This simple and useful expression now applies in the general case.

For example, we can test it at the two boundary cases: the top and bottom of the hypercube, which correctly gives 2*D (our double rule) and zero. To obtain the total number of possible subhypercubes from I to O we have to sum up the number produced at each level including the zeroth level itself. That is the total number of subhypercubes, T is given by:

$$T = 1 + \sum_{L=0}^{D} \{ \frac{2(D-L)}{L+1} S_L \}$$

and for D = 1, 2, 3, 4 ... we obtain T = 3, 9, 27, 81, ... Clearly the total subhypercubing multiplying factor = 3^D. This contrasts with the earlier quoted 4^D.

Several other factors intervene to reduce the upper limit of 3^D factor even further. How far is a function of certain assumptions made. What are these other factors? Well, for example, as far as "retaining subhypercubes for expansion at the next level" is concerned (we will return to this issue later) the "maximum" almost never occurs even for the most nonlinear of problems namely parity.

This problem dependency complicates the above theory. There is no easy relationship between the multiplying factor and the subhypercube multiplying factor only an upper bounded relationship. It is useful therefore to see what our practical work reveals. In a nutshell for H^2 to H^8 the multiplier is found to vary from 1 to 2 as we shall see shortly. Returning from our side-track we continue with the previous discussion.

... hence if back-propagation typically has a multiplier of 4 then it must still have considerable redundancy left in it even at H^2 to H^8. Can we do any better? The conclusion must be that if we keep an eye on the '**multiplier**' then it gives a good indication of the efficiency of the system.

There are a multitude of sources of inefficiency - some outside our control. Nevertheless for a later algorithm global0.8, which has eliminated most of them, we have the following results:

Figures 4.15 and 4.16 plot graphs for even parity the odd parity. Both odd and even parity problems were the basis of many runs for all dimensions from the 1st to the 13th. The two plots are overlaid in Figure 4.17 showing a considerable degree of similarity between both multipliers. In fact experience has shown that all aspects of the multipliers are indicative of the quality of the algorithm.

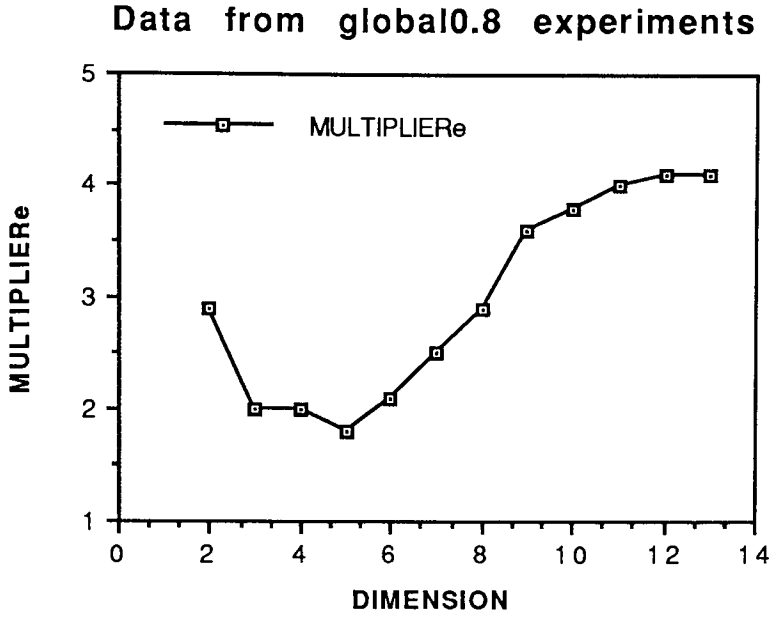


Figure 4.15 The even parity multipliers.

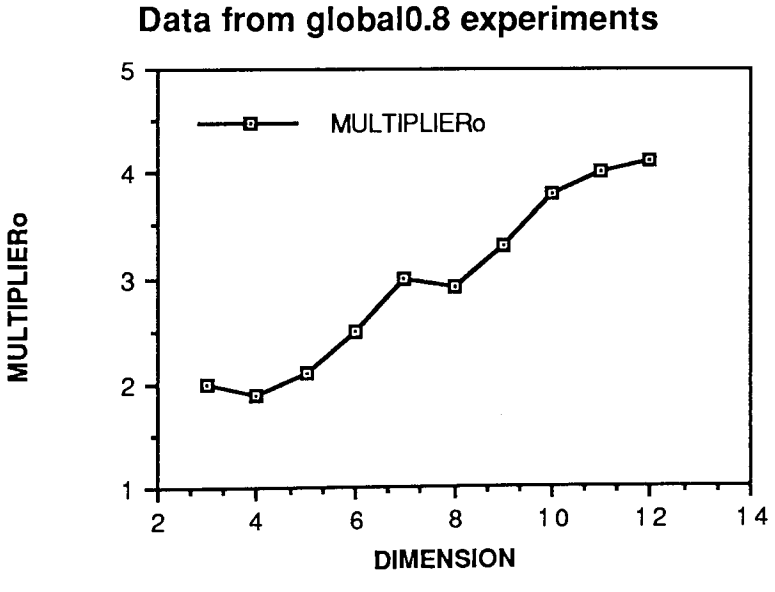


Figure 4.16 The odd parity multipliers.

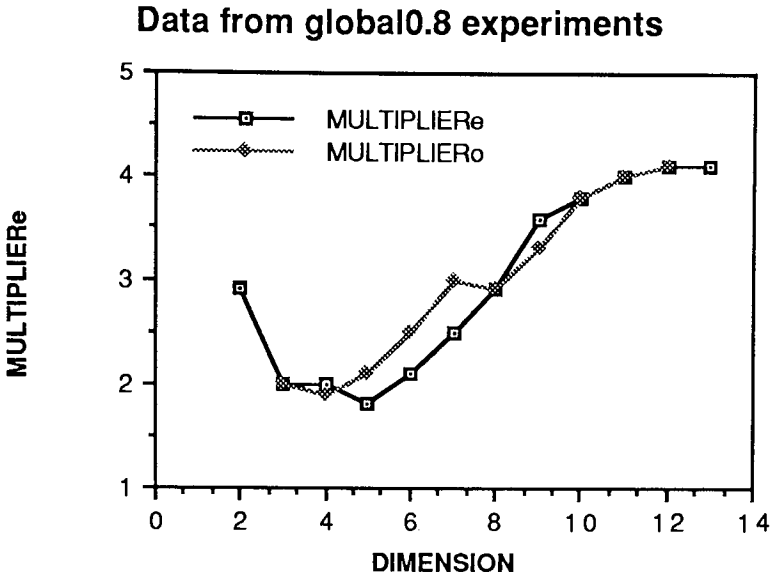


Figure 4.17 Comparing multipliers.

4.2.2.3 Results using learn.

As a contrast to all of the above graphs and in order to preview some of the effects of the algorithmic refinements, it is useful to consider similar sets of experiments for the final algorithm "learn".

One set of experiments which involved parity was carried out on the final version of the subhypercubing algorithm called 'learn'. These experiments were carried out on SPARCstation 1, server machine 2, after compilation. The parameters used (see later notes) were set at R = D, noise = 1, list formula = yes, debug = no, binary chop = true, Top_bot = false. Each of the subexperiments, experiment 1 to experiment 5 inclusive, involved timings for dimensions 1 to 13 for both odd and even parity, see Table 4.3.

These 130 results are plotted in Figure 4.18 - the uniformity of the results is noticeable. However the scale used is too small to show up individual variations - the icons used to plot expt1 to expt5 in fact overlap. Most points are indeed along the abscissa.

Consider the mean in Table 4.3 for both odd and even parity and instead plot the natural logarithm against the dimension, D. The graph given in Figure 4.19 is much more linear. There is some variation for the first one or two points on the graph. Caution should be exercised here since there is much higher degree of uncertainty in the timing in that region,

due to the fact that the times are of the same order as the resolution of the 'statistics' timing quanta, namely 17 msec. Other effects also occur here, for example there is the greatest ratio of overhead to algorithmic processing. Nevertheless the curve in this region appears to be real since it has been reproduced in many experiments and was also based upon the average of many runs. This logarithmic linearity, however, has a distinctive curve and is again indicative that the multiples are not a constant. This we shall explain later.

Expt.1	Expt.2	Expt.3	Expt.4	Expt.5	D	sum	mean	mult	ev/odd
0.000	0.000	0.000	0.000	0.017	1	0.017	0.0034		
0.000	0.000	0.000	0.160	0.000	1	0.160	0.0320		
0.050	0.050	0.033	0.033	0.050	2	0.216	0.0432		
0.050	0.034	0.043	0.033	0.050	2	0.210	0.0420		
0.083	0.084	0.116	0.116	0.100	3	0.499	0.0998	2.325	2.325
0.100	0.100	0.100	0.100	0.100	3	0.500	0.1000	2.380	2.380
0.200	0.200	0.183	0.183	0.200	4	0.966	0.1932	1.930	1.930
0.200	0.183	0.183	0.184	0.183	4	0.933	0.1866	1.870	1.870
0.383	0.350	0.383	0.383	0.367	5	1.866	0.3732	1.932	1.932
0.383	0.367	0.384	0.400	0.350	5	1.884	0.3768	2.016	2.016
0.767	0.767	0.766	0.767	0.767	6	3.834	0.7668	2.056	2.056
0.750	0.750	0.767	0.783	0.800	6	3.850	0.7700	2.042	2.042
1.866	1.900	1.883	1.883	1.884	7	9.416	1.8832	2.455	2.056
1.983	1.883	1.867	1.900	1.884	7	9.517	1.9034	2.471	2.471
5.516	5.617	5.550	5.567	5.550	8	27.800	5.5600	2.952	2.952
5.617	5.584	5.617	5.567	5.533	8	27.918	5.5836	2.934	2.934
19.417	19.35	19.300	19.450	19.700	9	97.217	19.4434	3.496	3.496
20.784	19.484	19.283	19.300	19.350	9	98.201	19.6402	3.517	3.517
75.250	75.183	74.416	74.433	74.433	10	373.715	74.7430	3.844	3.844
75.366	74.116	74.133	74.067	74.067	10	371.749	74.3498	3.785	3.785
302.417	299.767	298.633	296.633	296.433	11	1493.883	298.7766	3.997	3.997
303.667	299.167	296.350	296.250	298.533	11	1493.967	298.7934	4.018	4.018
1213.533	1207.050	1210.066	1212.517	1212.284	12	6055.450	1211.0900	4.053	4.053
1221.984	1212.267	1212.050	1212.533	1212.167	12	6071.001	1214.2002	4.063	4.063
5007.117	4977.733	4975.850	4977.233	4969.650	13	24907.583	4981.5166	4.113	4.113
4977.850	4977.233	5007.117	4975.733	4977.733	13	24915.666	4983.1332	4.104	4.104

Table 4.3 Learn multiplier results.

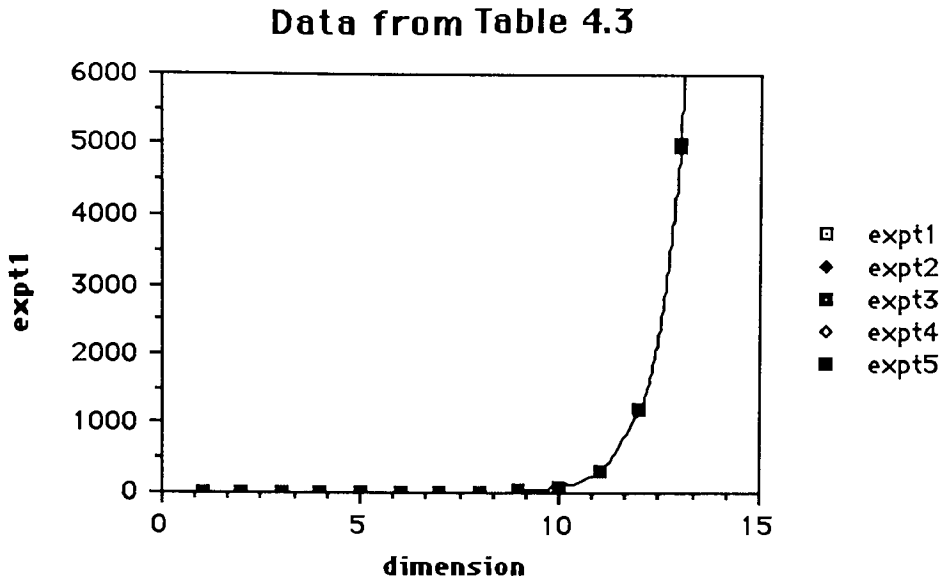


Figure 4.18 Exponential graph of learn multiplier results.

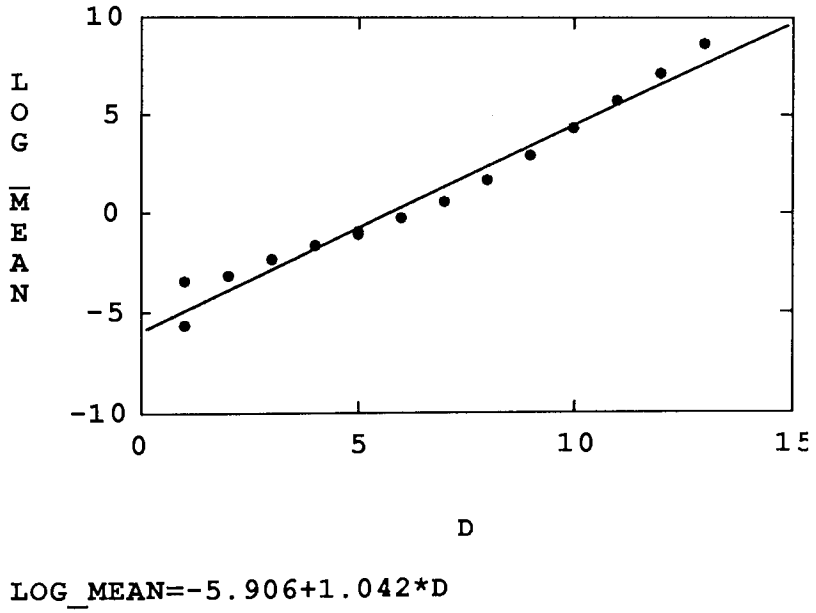


Figure 4.19 Logarithmic linearity of the explosion in learn.

It is useful to consider two checks:

1. What difference is there between the subexperiment runs as Figure 4.20? The answer is very little - this graph is remarkably linear.

2 What difference is there between the multipliers across all the experiments which are given for the individual odd and even parity cases, as Figure 4.21? Again we see a very linear graph. Learn is producing much more consistent results.

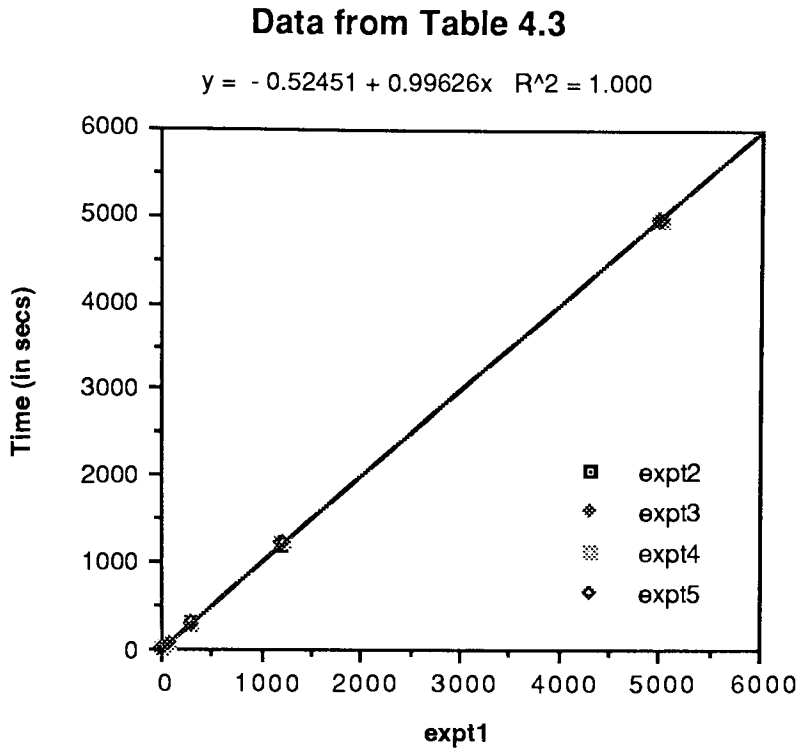


Figure 4.20 Comparing subexperiment runs in learn.

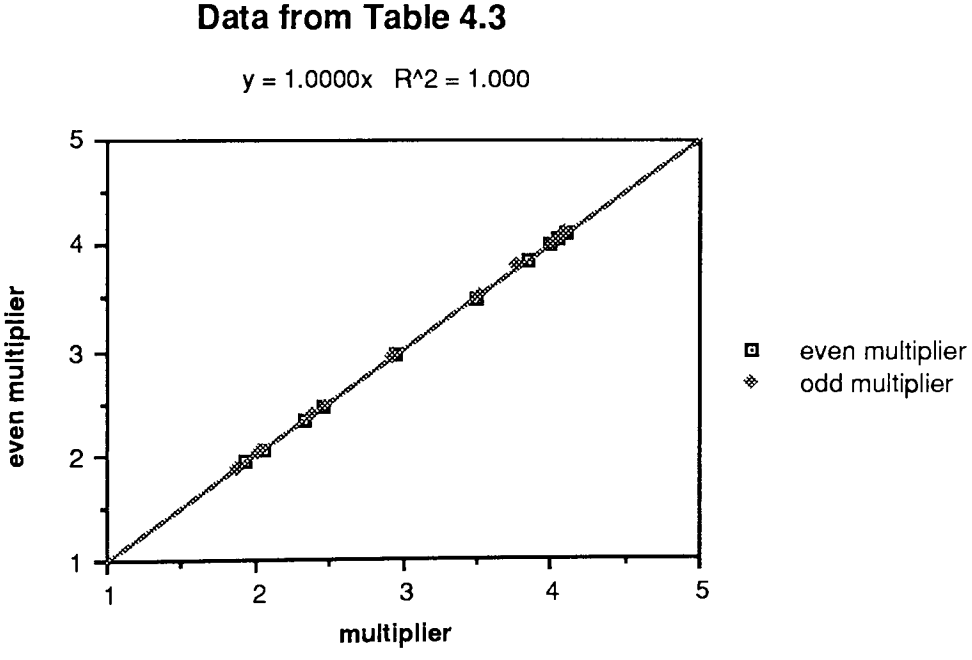


Figure 4.21 Comparing multipliers in learn.

It is further useful to plot the even multiplier against the dimension in Figure 4.22. The curve shows that the individual multipliers vary from about 2 to 4 over the dimension range. The mean is about 3 while the graph becomes asymptotic to about 4.25. A similar graph is plotted for odd parity in Figure 4.23.

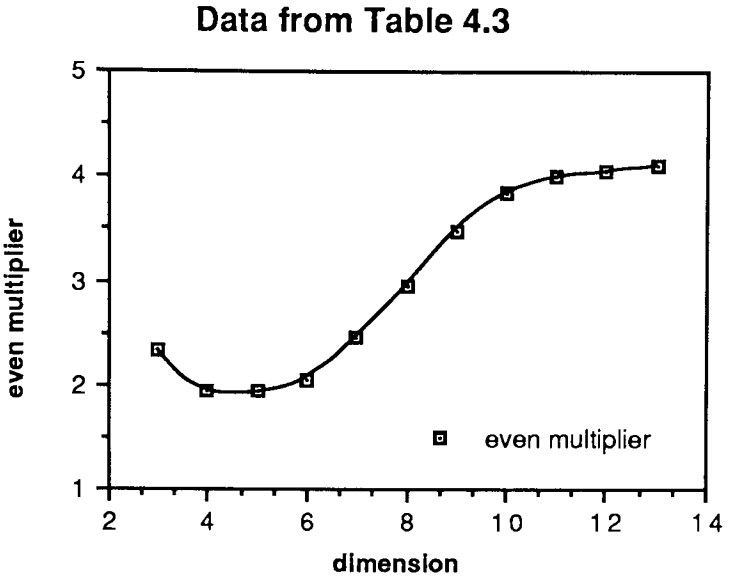


Figure 4.22 The even parity curve in learn.

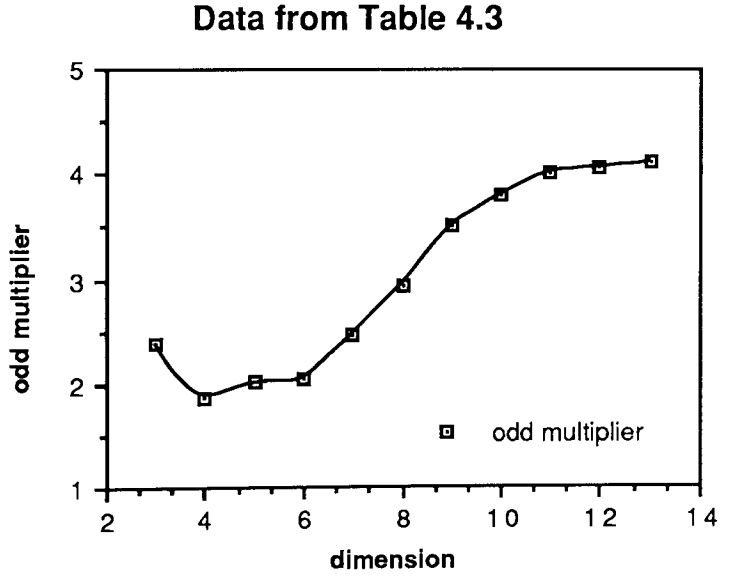


Figure 4.23 The odd parity curve in learn.

The algorithmic improvements are clearly seen in these curves when comparing with Figures 4.15 and 4.16, for example in the smoothness of the curves and the consistency between odd and even parity as seen in the overlap of Figure 4.24. In short as stated above,

learn is producing much more consistent results. Lastly the shape of Figure 4.24 is easily seen in Figure 4.19 although distorted by the log conversion and minus the first three points.

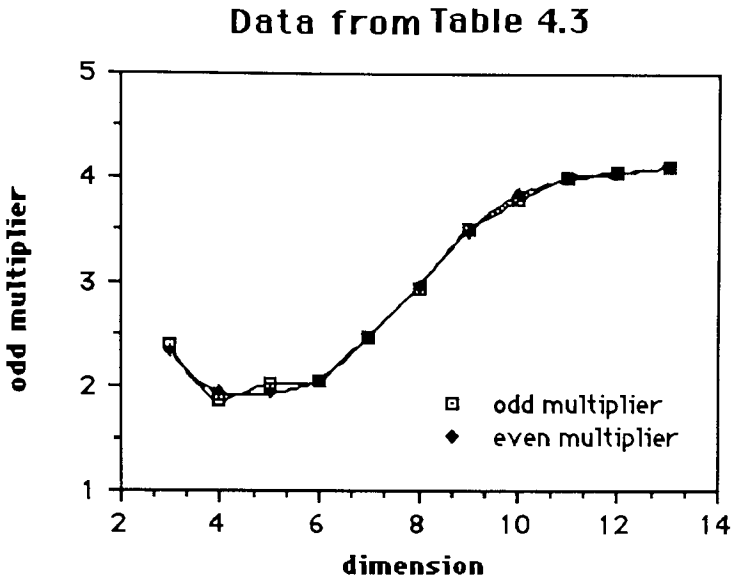


Figure 4.24 Comparing both odd and even multipliers in learn.

We have seen something of the hypercube system, the final learning algorithm, the sort of results and graphs obtained and a few illustrations of the some of the techniques used to refine the algorithm. We will now consider the development of the algorithm itself from the perspective of its basic concepts.

4.2.3 The global0 algorithm.

We begin by discussing the first version of the global approach of subhypercubing called "global0". This algorithm is based upon Michalski's "doublebeam" algorithm (Michalski, 1983). This first attempt is dissected in some reasonable detail in order to set the scene for further algorithmic variations. That said, we will only discuss the basic skeleton of the algorithm. Since Prolog is a very high-level language, at least as high as general pseudocode, we will at times, for clarity, state the relevant Prolog directly, since pseudocode would merely mislead. Various weaknesses in this first algorithm are also highlighted and compared with various refinements.

4.2.3.1 The dynamic databases.

The database of examples from the universe was initially bundled in with the code for testing purposes and only in later versions more flexibly separated out into a large set of independent files to be read in as required. These databases are generated by 'generators' and cross-checked by 'comparators' of various kinds. The connectionism of a node within the

hypercube is simply specified (according to the theory section 3.4.3.3) in the database in the format: **hypercube***(Index, Class)*. For example:

```
h([1,1,0,1],[1]).
h([1,0,0,0],[0]).
h([0,1,1,1],[1]).
h([0,0,1,1],[1]).
```

This particular example specifies positive and negative examples in H^4. Using positive logic, a class of "1" specifies a positive example, and a class of "0" specifies a negative example (by conventional standard in neural nets). The hypercube database and two others are declared "dynamic" for QUINTUS PROLOG by the following code:

```
:- dynamic h/2.
:- dynamic substack/1.
:- dynamic formula/1.
```

where ".../x". specifies the arity. The "substack" database has the unary relation form **substack***(Index)* and is able to utilise ϕ-booleans (section 3.4.3.2). In Prolog we indicate this variable by the underscore "_" in the Index list. One advantage of this notation is that it is possible to input *variables* in the exemplars. This is highly useful in a number of respects. For instance, we are then able to say in effect: " ... all examples of this type act in the following manner ..." This enormously reduces the number of examples that may otherwise be required to adequately define the space; leads to a very compact formulation of the result by the orthogonality of representation; and is a powerful facility which is *not* available, for example, in conventional neural nets. Since we have now mentioned the subject, let us also note that the "formula" database as a result of the machine is collected in the format: **formula***([<Subhypercube Index>])*.

4.2.3.2 The run module.

An initial runtime module (which calls the global algorithm) does certain external housekeeping tasks. This module is called "run" and it collects and prints out (fairly complicated) statistics on the time taken to run global0 in seconds to the nearest millisecond $\pm$ (17 $\pm$ 1) clocks (as in section 4.1.1.3).

4.2.3.3 The global module.

The global algorithm is called with the dimensionality as a parameter. A weakness of this algorithm was that the dimension was not automatically asked for and had to be manually inserted in early versions of the algorithm such as global0. We distinguish between

the terms "learning" and "non-learning". (These are not particularly enlightening terms). They refer to the case where there does or does not exist sparsity, respectively, within the hypercube.

Again a weakness was that it would have been better to ask if "learning" or "non-learning" is required automatically and then instantiate the dynamic database in order to save the later irrelevant message of "ERROR unless learning ..." invoked by the later procedure "runcase2". This modification together with a change in the "runcase" modules was done in later versions. The code for the global procedure, of format **global***(Dimension),* is as follows.

```
global(D) :-
        length(Subhcube,D),
        doublebeamsearch(true,[Subhcube]),
        exitglobal,                                  % iff non-incremental!
        elimstack,
        write('Formula is  '), nl,
        listingformula.
```

The clause **length***(Subhcube, D)* is a beautiful example of the elegance of the **declarative style of Prolog.** In one line of code this very compactly produces the hypercube for any dimension. For example, for dimension D = 2 we immediately produce the hypercube as a "zeroth level subhypercube" (section 3.4.3) in ϕ-boolean terms as [_,_]. (Note: with respect to parameters, an underscore and/or a capital letter at the start of a word implies a variable to Prolog as section 4.2.1.1).

Next there is a call to the doublebeam searching algorithm, "doublebeamsearch". In this algorithm the doublebeam is created, searched and manipulated by the use of stacks. The parameter "true" in this clause is an initialiser ensuring that [Subhcube] is taken to be "Stack1" in the second "doublebeamsearch" clause.

A problem with the use of the list of lists parameter [Subhcube] is that this approach restricts the algorithm to dimensions of a given sparsity where there are < 112 subhypercubes at any given subhypercube depth. (The problem being due to the version of QUINTUS PROLOG used at the time which could not manage list lengths greater than this. Later versions did not have this restriction but it provided early trouble for the development of our algorithm).

On the other hand list manipulation is three times faster than database access which gave a confusing initial advantage to early algorithms, until we spotted the point. At the cross-over between the two (global0.4 and global0.5) the list usage is seen to enable faster speeds but only at the lower dimensions, as Figure 4.25. After which other algorithmic refinements come to the fore.

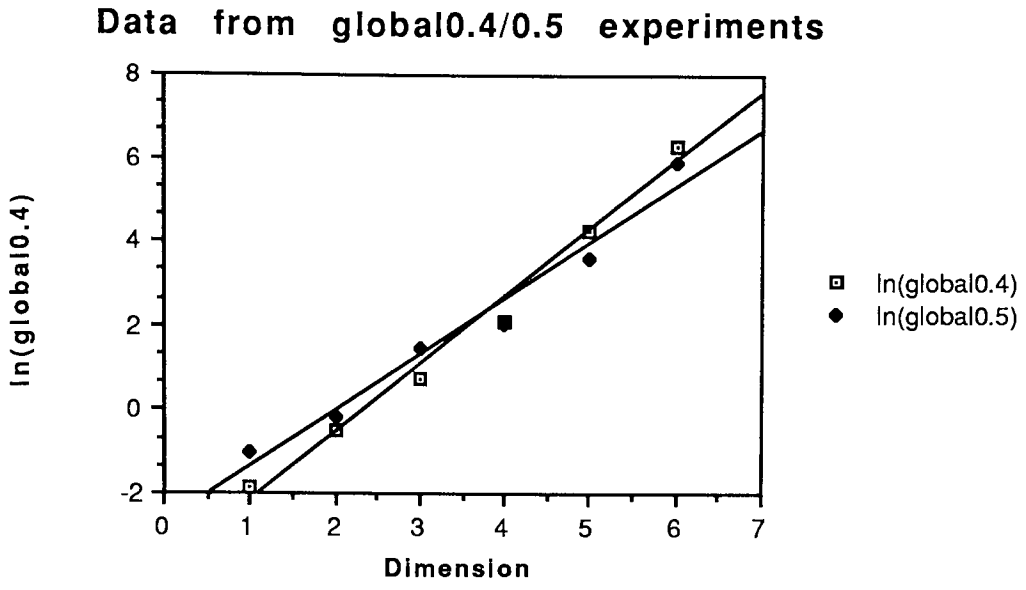


Figure 4.25 List/database speed comparison.

But there are several further difficulties. For example, the algorithm runs approximately twice as fast by the sixth dimension (when any database redundancy, duplication, etc. is starting to become significant) by halving the database size due to the overhead it incurs. This is illustrated in Figure 4.26. This is a QUINTUS environment effect. A more complex environmental effect - due to the double network servers' swop-in time (server UHURA to server Castor, server Castor to SPARCstation server) is that the first two runs have to be discounted as they are very slow. Thereafter the times are much more constant:

e.g 1.917, 1.634, 1.500, 1.517, 1,500, ...

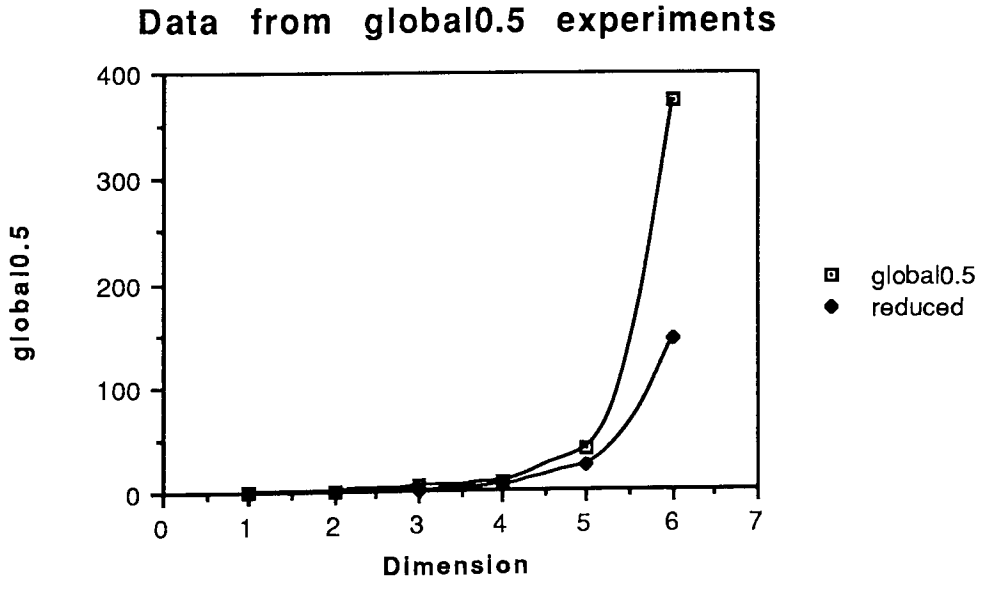


Figure 4.26 Effect of reducing the database size.

4.2.3.3.1 Pascal's triangle.

In the theory (section 3.4.5), we referred to the usefulness of Pascal's triangle. An arrangement of Pascal's triangle gives the number of unique subhypercubes at any given subhypercube level, as Table 4.4. To obtain the required number of hyperfaces it is only necessary to multiply the number in the appropriate cell by the number in the bottom row (which is 2^n where n is the dimension) obtained via the diagonal. For example, the number of faces on H^7 is n = 7 (1st column) -> 21 (faces column) by 32 (bottom line along diagonal), hence:

hyperfaces$(H^7, Faces)$ = 672.

n	Nodes H^0	Edges H^1	Faces H^2	Cubes H^3	4-cubes H^4	5-cubes H^5	6-cubes H^6	7-cubes H^7	8-cubes H^8
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	
8	1	8	28	70	70	56	28	8	1
	256	128	64	32	16	8	4	2	1

Table 4.4 Pascal's triangle of unique subhypercubes.

A great deal of further useful information can be obtained from Table 4.4. For example, we can predict the number of subhypercube levels, number of nodes per level, we can generate the correct number of nodes in the next level, the number of nodes in the maximum node level, etc. For example, the number of nodes at each level of H^7, can be read off directly along the row for n = 7 as: 1,7,21,35,35,21,7,1; the maximum node level has 35 nodes; 35 is obtained by adding the above pair in Table 4.4, that is, directly above (20) and above along the diagonal (15); etc. Note that the multiplier for hyperfaces gives the maximum possible number of positive (class 1) subhypercubes at that level. In practice, this number will typically be very much smaller than the maximum, due mainly to sparsity, but also to negative exemplars.

4.2.3.3.2 Unique subhypercubes.

Our arrangement of Pascal's triangle gives the maximum number of unique subhypercubes at any given subhypercube level. Bearing in mind the above restrictions on the list length, there are two problems. Firstly, the necessity of ensuring uniqueness is obvious once seen, but subtly opaque until 'the penny drops'. The theory, of course, assumes uniqueness but this 'hidden variable' (section 3.1) was not explicit in the theory. Should the algorithm fail to produce a **unique set of subhypercubes**, then the number of subhypercubes considered will, in the worst case, be greater than the theoretical value and may well exceed our maximum list length. This subtle point has a far from easy solution and was problematic in early versions of the algorithm causing an unnecessary further explosion in the search times.

We can compare graphically the first version which solved the uniqueness problem, global0.4 with its immediate predecessor, global0.3 in Figure 4.27. Notice that the extra processing required meant that initially the algorithm was slower - an excellent reason for not making assumptions based upon only the first few results! This cross-over pattern happened quite often. The run timings used for Figure 4.27 were *not* based on the parity problem but were the typical times resulting from averaging across all the different problem types (including parity) within that dimension and for that algorithmic version in all being 164 tests. It is useful to compare the average or "typical case" and the "worst case" or parity. See Figure 4.28. The typical case rapidly becomes an insignificantly small percentage of the "worst case" (one reason why we benchmark parity), as is seen, parity explodes much faster. E.g. even parity for D = 3 => 2.117 secs., whereas even parity for D = 6 => 641.666 secs. Incidentally, the "best case" is an almost horizontal line independent of dimension at 0.034 seconds.

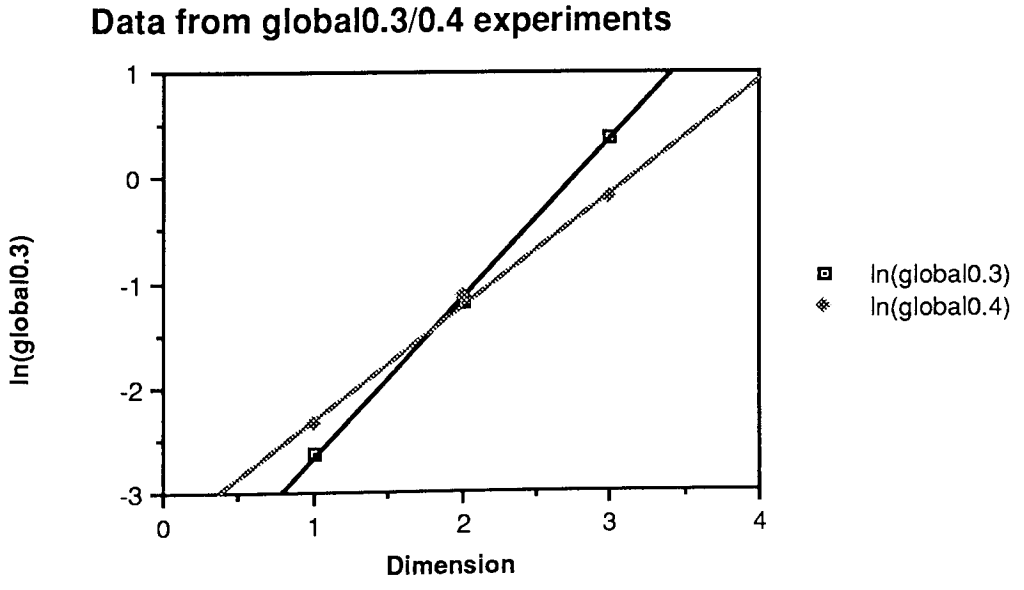


Figure 4.27 Uniqueness comparison.

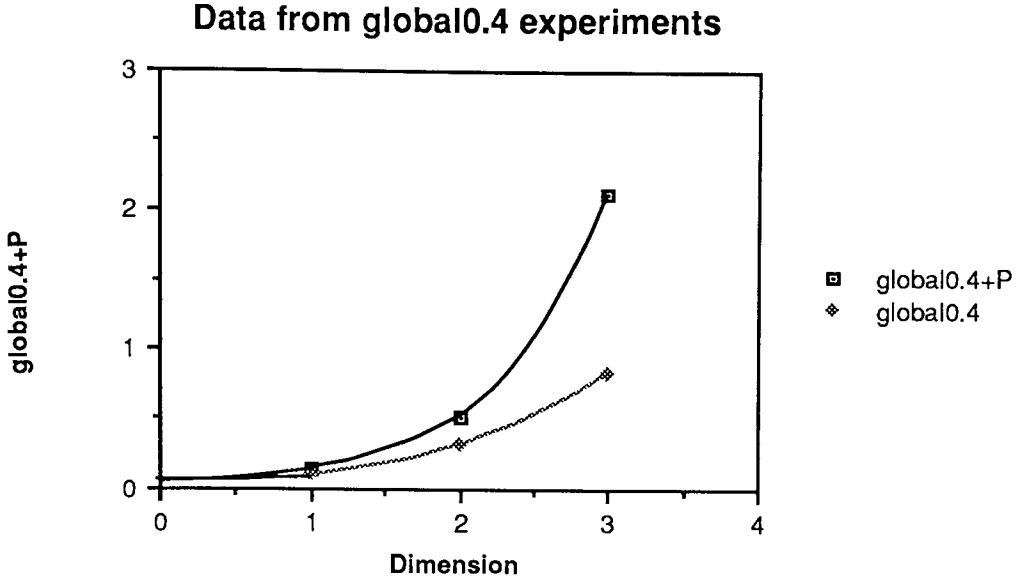


Figure 4.28 Typical/parity comparison.

4.2.3.3.3 The list length problem.

Secondly, what happens if we assume uniqueness and a maximum list length of 112? Briefly, in the worst case, we are limited to a maximum of the 5th dimension only! This is *very* limiting and, until we found the solution, *depressingly disappointing*. Further, without uniqueness, we are limited to the 4th, 3rd, ... dimension, depending upon the redundancy (lack of uniqueness), which is catastrophic. In the style of Modern QUINTUS PROLOG, the "preferred solution" is via a deep modification to eliminate the stack of subhypercubes and store all subhypercubes dynamically in yet another database. [6] This, as we found later, has its own explosive overhead, particularly apparent for larger problems even if the algorithm itself were to have no explosion whatsoever (section 4.2.3.3). The problem is illustrated in Table 4.5. This a known problem with QUINTUS PROLOG. Searching a large database takes a significant time. The problem has also become worse with later versions of QUINTUS - in the manuals they say it is due to certain (unspecified) trade-offs they have had to make in providing new facilities. The database access times were shown by iterative database access experiments to have consistency as Table 4.5. (The two uppermost rows were extrapolated from the measured lower rows).

---

6 This may well be "in the style of" the "preferred/recommended modern approach to QUINTUS PROLOG." It is perhaps good sales chat, but we have technical objections. For example, being almost forced to use Quintus in this way, we observe that it makes excessive use of a common global data area, as in the old static allocation languages. These Quintus globals (not to be confused with our "global" algorithm) are easy to use, even easier to use badly and thus very much against the trend of modern software engineering which needs to be applied even more, not less, vigorously and attentively in AI due to its inherent complexity.

Database accesses	Time taken (in secs)	Dimension
1	0.0025	1
10	0.025	4
100	0.25	7
1,000	2.5	11
10,000	25	14
100,000	250	17

Table 4.5 Dynamic database search overhead.

Fortunately, this is a once off overhead for a given predicate. Nevertheless, it is useful to see the problem at its worst (which is most unlikely to happen in practice). Given an actual minimal overhead of 0.1 secs then for H^{17} we have a total time T of:

$$T = 100{,}000 * (250 + 0.1)$$
$$= 289.47 \text{ days}$$

In practice, experience has shown that the worst case is likely to be of the order of: (Time taken) * (number of levels). Hence for this example we have 250 * 18 = 1.25 hours. This is an annoying excess to the algorithm, since it is of the same order of magnitude as the algorithmic time taken for the later fast versions of the algorithm. It also introduces an unwarranted further explosion in time which becomes apparent at higher dimensions. Perhaps QUINTUS will ease the problem since other researchers have found it equally annoying.

Summarising, in order to avoid the severe list length restriction the proposed modification is to store subhypercubes dynamically in the database. This has the advantage that there is no restriction on the dimensionality of the problem considered (we have tested up to D = 1040), but it introduces a new disadvantage in the form of a second source of exponentially compounding speed loss due to increasing database access times as the dimensionality increases. There are several further points.

4.2.3.3.4 Further aspects.

Firstly, there is another problem with the code so far. There is no attempt at the outset to detect zero level subhypercubes (section 3.4.3). As stated in section 3.4.3 there are only ever two possible zero level subhypercubes, the complete hypercube contains only positive exemplars (or I), and the complete hypercube contains only negative exemplars (or O). Granted this is trivial and would not presumably occur in practice except by error. Nonetheless, for the sake of completeness, rigour and good precautionary programming we ought to first check for this "all or none" case before considering the subhypercube levels

within doublebeamsearch. If I or O (section 3.4.3) occurs then the formula procedure will output nothing, which is incorrect. Secondly, there is no check on the correlation between the size of database used and the invoked dimensionality - clearly a possible source of error.

Thirdly, the class can be used to assign a marker to indicate that connection has been visited, thereby avoiding redundant work. There is a problem with incremental learning in that care must be taken to ensure that these markers have been reset in the database for the next run. The later point is taken care of in the 'exitglobal' procedure which recursively resets the markers.

This might seem unnecessary - far from it. This procedure is still insufficient. The following point holds for all dynamic databases. Using the various facilities of QUINTUS PROLOG, for example the "trace mode," it is possible to abandon the program before the natural end of the program. In this situation the dynamic databases do not get reset. Even if the program is recompiled, the next run will begin in an erroneous state. Analogous comments apply for the next clause "elimstack" which similarly recursively eliminates the dynamic substack.

The last clause in global is "listingformula". This produces the formula in the general case as a stack of ϕ-boolean lists such as:

[0,_,1,_]
[1,_,_,0]

This again avoids the 112 list length restriction. Output in this format (as opposed to a list of lists) is easy to scan visually, particularly for long complex formulae.

4.2.3.4 The doublebeamsearch stack cycling machine.

A simple introduction to the ideas behind the double beam concept is illustrated below in Figure 4.29. Subhypercube 'A' is 'incomplete' since it contains both 1's and 0's. Subhypercube 'B' is 'complete'. If in a single beam search, accessing is in the order:

..., A, ..., B, ...

then 'A' will be expanded unnecessarily. (The interhyperface between 'A' and 'B' contains the only 'complete' subhypercube of hyperface 'A' and this interhyperface is a subset of the already 'complete' subhypercube 'B' - hence entailing redundant information and failing to produce the simplest possible explanation of the data).

Predicting the ordering is an exponential exercise for increasing dimensionality, as is reordering. Further, remembering that these hyperfaces may themselves be of very high dimensionality, the unnecessary additional work is also exponentially explosive.

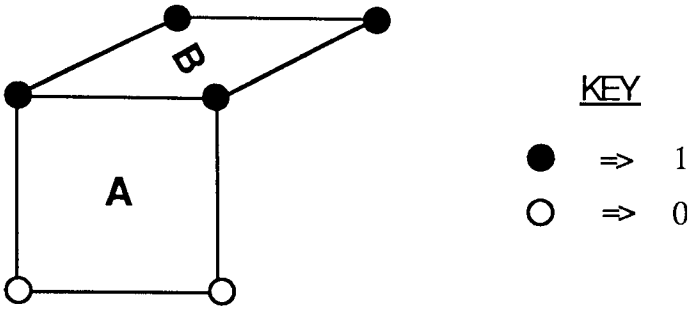


Figure 4.29 Interhyperface algorithmic significance.

In contrast, with a double beam search algorithm the complete set of subhypercubes at that level is processed twice. In the above case, if 'A' is accessed first and found to be 'incomplete' then it can be flagged as such. Later 'B' is found to be complete, hence the class 1's in the interhyperface become x's. On the second beam search it is seen that some subhypercubes need not be expanded, for example 'A', since it no longer contains either only 1's or solely both 1's and x's. In general, the very probable expense of many recursive expansions is thereby avoided.

It might be thought that the solution lies in the intermediate possibility, in this case {..., A, B, recheck A, ...} to see if it is still necessary to expand it. Many problems arise with this approach. We mention one with several consequences, namely, considering that in general there will be a large number of subhypercubes to be processed, finding their interhyperfaces requires expansion at the next level, possibly unnecessarily, throws away information possibly of value later or keeps possibly unnecessary information and is exponential and therefore explosive anyway. Interhyperfaces between adjacent subhypercubes explosively form ever more complex combinations of interhyperfaces. At the very least we compare each of n subhypercubes with the n - 1 others to find those of common interhyperfaces (some of which may previously have been accounted for) which then require previewing at the next level!

Because there are very many more subtle combinations than the very simple example in Figure 4.29, the conceptual difficulties become severe with increasing dimensionality. To save space and very difficult descriptions, we merely note that the hypercube is an orthogonal representation. Hence a trend is perceived - after the combinations of two hypercube dimensions (odd and even) - namely, that we require a 'doublebeam' algorithm.

'Doublebeamsearch' is a stack cycling machine, see Figure 4.30. Due to the list length problem the stacks should be dynamic. Processing cycles through (subhypercube) level n and its subhypercube level n + 1 and then repeats thereby creating a recursive spiral

of double beam searches. Within each cycle there are two beam searches, firstbeam and secondbeam, processing stacks accordingly. Each of these searches across the full beam of the subhypercubes at that level.

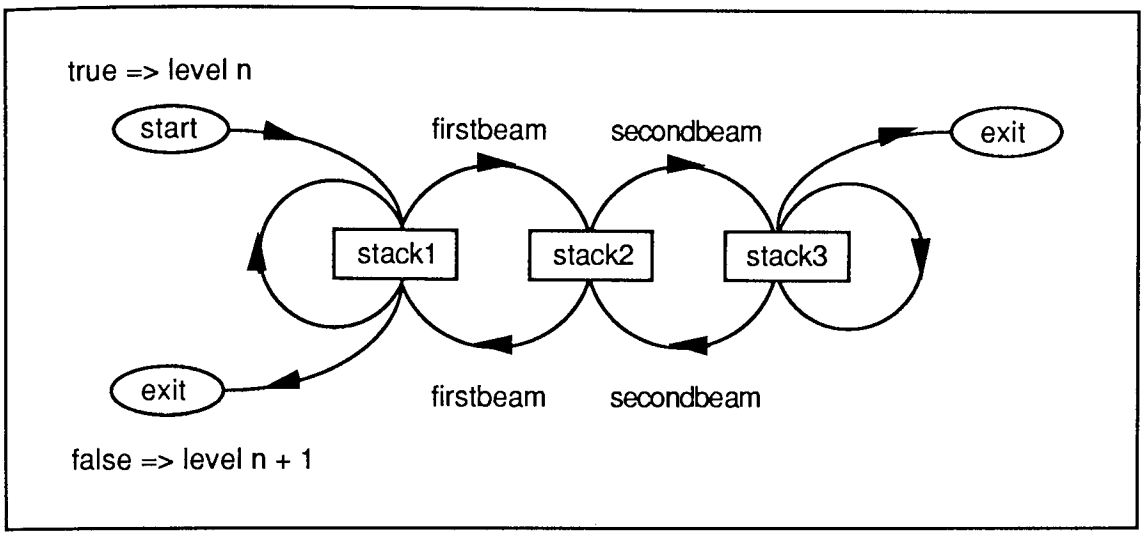


Figure 4.30 The double beam search stack cycling machine.

It is assumed that neither firstbeam nor secondbeam will ever fail. The recursion unfolds back up through the doublebeamsearch exit clause. The code for doublebeamsearch of format **doublebeamsearch***(Beam_indicator, Stack)* is given below:

```
doublebeamsearch(_,[]).
doublebeamsearch(true,Stack1) :-
        firstbeam(Stack1,Stack2),
        secondbeam(Stack2,Stack3),
        doublebeamsearch(false,Stack3).
doublebeamsearch(false,Stack3) :-
        firstbeam(Stack3,Stack2),
        secondbeam(Stack2,Stack1),
        doublebeamsearch(true,Stack1).
```

The first parameter (either true or false) selects the level. The first clause exits on the condition of the empty stack, that is, when there are no more subhypercubes to search.

4.2.3.5 The firstbeam procedure.

In firstbeam, of format **firstbeam***(First_Stack, Second_Stack)* as below, the first clause gives the general case for stack1 or stack3, level dependent, containing several indices, one per subhypercube. The predicate testsubhypercube1 tests the topmost index X, returning Y which is either [Top] from the later "runcase" (where 'Top' is X implying expand the subhypercube or [[]] indicating no need to expand the equivalent subhypercube). Doconc is a special type of concatenation procedure.

```
firstbeam([X|R],Expander) :-
        testsubhcube1(X,Y),
        firstbeam(R,S),
        doconc(Y,S,Expander).
firstbeam([],[]).
```

One major problem with QUINTUS PROLOG is that although the language assumes that the user will write recursive based code, recursion being an integral part of the language, in practice this is problematic. The QUINTUS manuals warn against anything other than minimal *nested* use of recursion (at best to depths of 5 only). The penalty otherwise is excessive system time usage in enormous garbage collection exercises. [7]

Clearly this practical restriction is problem dependent. For our purposes we are highly likely to exceed this limit due to:

a) the program being inherently recursive,
b) larger hypercubes forcing further recursion.

4.2.3.6 Necessary iterative complexity.

The recommended Prolog solution when faced with this problem, is to use *iteration* rather than recursion. We could say use iteration for doublebeamsearch. (In practice it was later found to be essential for all recursive modules within the program). Unfortunately, iteration in Prolog produces - however hard one tries to follow the tenets of software engineering - *very opaque code*. This is due to the fact that iteration is not essentially part of the language - its afterthought nature resulting in a software engineer's nightmare. There are four problems.

1) Iteration necessitates complicating the structure of the module.
2) Iteration is so difficult to produce in Prolog that it results in very opaque code.
3) The declarative attractions of the language are sacrificed to (an iterative) procedural style in order to gain acceptable speed.
4) All this implies grave limitations for the use of Prolog (a) in larger research projects where time is likely to be critical (such as for larger hypercubes) and (b) becoming a commercially acceptable language.

We know of no serious use of QUINTUS PROLOG which has not had to resort to iteration. So the problem is not rare. There are sadly several other known problems in Prolog such as "red cuts" (Campbell, 1985; Bratko, 1990). In short, our experience with the language could be summarised as follows:

---

7 It appears that the total preceding environment has to be copied into the Prolog system stack each time recursion occurs. Memory demands/requirements then become quite unreasonable - very rapidly reaching the gigabyte region. (See Campbell, 1985).

The marvellous benefits and beauty of Prolog are precisely balanced by its unacceptable shortcomings and enforced ugliness.

The reader should not misunderstand our intent. The above comments refer particularly to a perceived development rather than research phase for the present and other products. In short, our experience in industry mitigates against recommending Prolog for engineering or commercial usage. For research purposes we are still impressed with the language.

4.2.3.7 The testsubhcube rule.

The testsubhcube rule is as follows:

```
testsubhcube1(Top,Expand) :-
        copy_term(Top,Top1),
        testcase1(Top1,Case1),
        copy_term(Top,Top0),
        testcase0(Top0,Case0),
        copy_term(Top,Topx),
        testcasex(Topx,Casex),
        runcase1(Top,[Case1,Case0,Casex],Expand).
```

The testcase modules test the virtual database. The Cases formed are classes. For example, Case1 means it is true/false that the subhypercube contains class 1. Cases are collected together as a "profile list" of the subhypercube in question for further processing in runcase1. For example, [true, false, true] implies that there are one or more examples of a connection within the subhypercube of class = 1 and class = x (marked), but no examples of class = 0. Upon querying the database with the binary testcase relations, the 'Top' ϕ-boolean masks become instantiated with the relevant subhypercube, thereby invalidating the next database access. Our solution to this tricky problem was the "copy_term" relation. The meta-logical predicate **copy_term***(Term, Copy)* makes a copy of its first argument in which all variables have been replaced by new variables which occur nowhere else as defined and **"unified"** in the rule:

```
copy_term(Term, Copy) :-
        recorda(copy, copy(Term), DBref),
        instance(DBref, copy(Temp)),
        erase(DBref),
        Copy = Temp.
```

4.2.3.8 The runcase1 clauses.

The profile list "Profilelist" has eight possibilities. These 8 possibilities are catered for by the 8 clause alternatives of runcase1. This is equivalent to a "case selector" in say ADA, or a series of "if then else" statements in a conventional block structured language, as

for example, is used in a lexical analyser. Runcase is called with three parameters: **runcase1***(Subhypercube, Profilelist, Expansion).*

The runcase clauses are as follows:

```
runcase1(_,[false,false,false],[[]]).          % none!
runcase1(_,[false,false,true],[[]]).           % all x's
runcase1(_,[false,true,false],[[]]).           % all 0's
runcase1(_,[false,true,true],[[]]).            % all 0's & x's
runcase1(Top,[true,false,false],[[]]) :-       % all 1's
        asserta(formula(Top)),
        copy_term(Top,Top1),
        convertclass(Top1).
runcase1(Top,[true,false,true],[[]]) :-        % 1's & x's
        asserta(formula(Top)),
        copy_term(Top,Top1),
        convertclass(Top1).
runcase1(Top,[true,true,false],[Top]).         % 1's & 0's
runcase1(Top,[true,true,true],[Top]).          % 1's, 0's, x's
```

The item for expansion is returned as the object "Expansion" for ultimate doconc processing in firstmode and thence in doublebeamsearch to create Stack2 for the secondbeam procedure. Some assumptions are made. If examination of the subhypercube results in all 1's or all 1's and x's then the subhypercube is orthogonally assumed to be **"complete".** Sparsity means that some connections are still unknown. Since there is *no evidence against* the subhypercube being "complete" we assume it to be *complete as far as is known.* For example consider Figure 4.31.

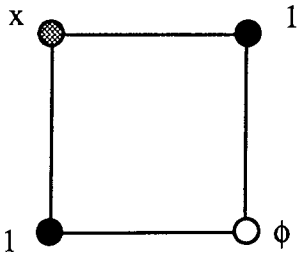


Figure 4.31 Subhypercube completion.

The values in Figure 4.31 refer to the class. There are two 1's, one 'x' and one unknown. The 'x' results from previous processing of another subhypercube which shares a hyperface with the subhypercube in Figure 4.31. Since only 1 -> x , the x refers to a known '1'. It follows that the database originally contained three one's and one (virtual) unknown as binary relations h. With no evidence to the contrary we assume the unknown is also a '1'. This completes the subhypercube.

If we are to abide by the search for the simplest possible explanation of the data (as in discussed in detail in sections 3.3.3, 3.4.4.3, 3.4.5.3, 3.5.8 and 3.5.8.3) then, clearly,

the simplest possible mathematical explanation is to complete the subhypercube and assume it is all 1's. (There is no way the sum of diverse parts can have a more compact mathematical expression than the whole since we are considering associative hypercubes).

Similarly one or more unknowns in a subhypercube containing only 0's are assumed ignorable, because there is no evidence to suggest other than that the whole subhypercube has class = 0. If the subhypercube contains all x's or 0's and x's there is no new positive exemplar information *not already taken account of* to suggest other than that the subhypercube may be regarded as all 0's. (To do so would be to invite duplication of effort at some lower level).

In Figure 4.32 (a) the subhypercube was originally seen to contain a '1' and a '0', hence its subhypercubes must be expanded. (It is *most efficient* to test for only **one** such pair, *whatever the size of the subhypercube,* in order to ensure the requisite necessity for expansion. This neatly fights the exponential explosion. It is one of our major techniques. Indeed to capitalise on this advantage we attempt to reapply the same approach throughout the algorithm - but with only partial success).

Subhypercube 'A' in Figure 4.32 has an 'x' and a '1', hence we take it as "complete" and of class '1'. If, however, we then assume x = 1 then subhypercube 'B' is taken as "complete" leading to Figure 4.32 (b), which then has a bias towards positive information. We are, however, at pains to avoid this (section 3.3.2).

Hence we assume that an 'x' means that the information has already been taken account of. Subhypercube 'D' is then taken as class '0' and since 'A' is of class '1', there is nothing further to do - 'B' and 'C' are not processed. The result is the unbiased situation according to (section 3.3.2) as in Figure 4.32 (c). Therefore, in order to abide by (section 3.3.2) the cases of all 'x', all 0 and all 0's and x's are taken as subhypercubes of class '0'. The case of all unknowns (section 3.4.3.2) within the subhypercube, in accordance with (section 3.3.2) is taken as class '0'.

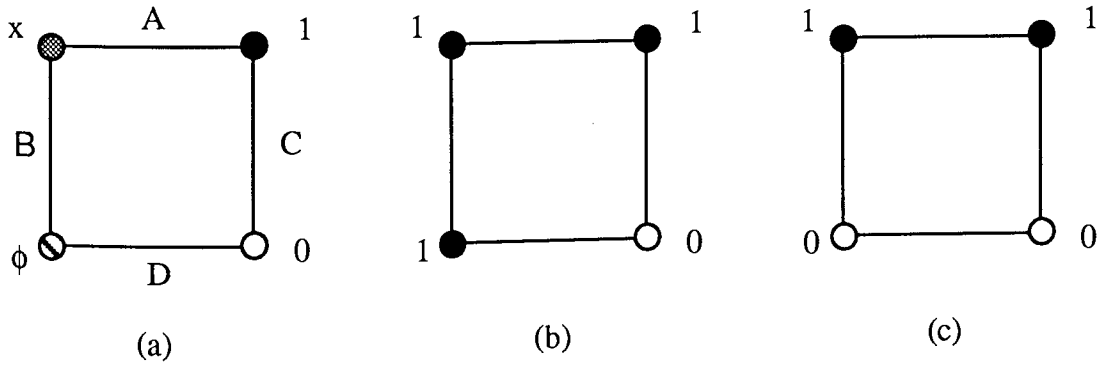


Figure 4.32 Considering ϕ-boolean completion.

The above resolution is based upon a "weight of evidence" argument for assumptions. Clearly also, all this attempts to enact the theory of chapter 3.

"Convertclass" converts the class of connections within the subhypercube considered by "runcase" in the case of classification '1' to 'x'. This is an expensive process implying recursive conversion of **all** 1's within the subhypercube to x's. So on obtaining the formula we convertclass.

When should we reconvert back again? Some possibilities are: after the next call to that subhypercube, at the end of the beam, at the end of the level, on exiting doublebeamsearch, at the end of the interactive use of that dimension. Consider Figure 4.33 where there are two abutting subhypercubes.

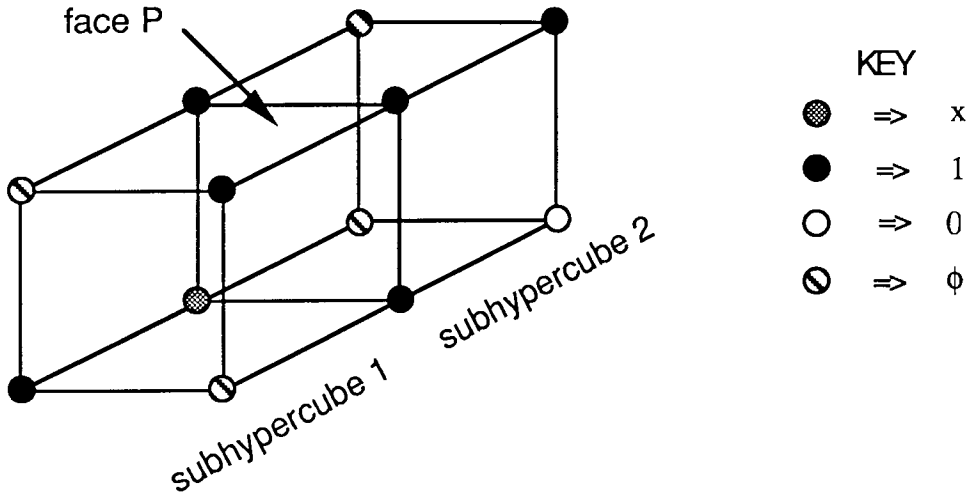


Figure 4.33 Considering two subhypercubes.

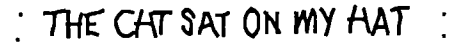
Subhypercube 1 is processed leading to formula 1. Subhypercube 2 contains 1's and 0's and is therefore expanded. If all x's -> 1's at the start of that beam then face P, as a subhypercube of subhypercube 2 will reduce to formula 2 because it is 'complete' in 1's. The problem is that formula 2 is then a subset of formula 1. Therefore, if on obtaining a formula we convert 1 -> x and reconvert either at the end of the beam or level then there is a failure to produce minimum polynomials hence breaking our requirement in section 3.4.5. Notice that this algorithm does not distinguish by markers between a connection having been visited and a connection having been used to create a formula upon completion. We will gloss over the many possible complexities, considerations and possible algorithms resulting from this and similar considerations, since these algorithms were ultimately discarded.

A second assumption is the more subtle one. We assume that, in order to reflect the universe, the data has a reasonable spread across the universe (Huyser & Horowitz, 1988).

To obtain this it is necessary that the data exists to at least pin-point the major hyperfaces of the problem. In obtaining a reasonable spread of data in neural nets the assumption is usually that at least three times the dimensionality of exemplars are required; four times would be preferable. [8]

If we employ the same rule, then for dimension D = 20 we need 60 to 80 examples spread across the hypercube space, not necessarily evenly if uneven distribution highlights the problem more accurately. If we cannot make this assumption or if sufficient information is available then we have the "non-learning" case (section 4.2.1.3) where there are 2^D exemplars. In this case, particularly for smaller D, less than D_{max} (where D_{max} is for the present hardware, environment, and typical problem, etc. about 20), the subhypercube algorithm acts an **information reducer.** [9] For example, rather than have to store a table with say a million results, the table can be compacted into a few formulae from which the result for any given case can be regenerated. There are many situations where such a machine could prove useful.

In the alternative case of "learning" (section 4.2.1.3) processing may be taken to the ultimate necessary subhypercube level, typically halfway down the heterarchy (section 3.4.5.3). In this case D would again need to be less than D_{max}, or it could be very much higher $D >> D_{max}$ if 100% accuracy is not required. The later implies **resolution** where the result is only approximate. Very often we find approximation is sufficient. For example, the child's handwritten complete sentence:



may well have been read as "THE CAT SAT ON MY HAT". A closer examination of the letters reveals two which are identical and yet represent different letters, namely the A and H of cat and hat. Were we to completely process the sentence then these two letters would have been undecidable. Clearly we rely upon partial processing of letters, words, semantics, etc. and combine approximate answers to obtain the result contextually. Similarly, approximate results may often be sufficient for larger problems. In this regard, depending upon the error tolerated the hypercube is able to process up to very large problems indeed e.g. D = 68 has been verified.

---

8 We ignore here our timely variables' advantage, which may well considerably reduce this calculation as section 4.2.1.1.

9 D_{max} is a variable. The actual value depends upon (1) how long processing may take (or we are prepared to wait for) e.g. secs, mins, hours, etc., (2) the speed of the machine (hardware, disc access time, etc.), (3) the software speed (language, windowing environment, operating system, etc.), (4) the extent of processing parallelism, (5) technological advances, for example, in hardware logic, mips, disc access speed (it is noteworthy that the law of processor speed-up has been approximately linear since microprocessor inception at 10 times speed-up every eight years). The combination of these points could well change a limit of D = 20 to D = 40 plus. Typical problems have a dimensionality of about 25 (as do actual neurons - a nice matching!). Problems greater than D = 40 are rare.

4.2.3.9 Secondbeam.

The secondbeam module is initially similar to firstbeam together with some "precautionary programming" with respect to firstbeam. If there are both 1's and 0's in a subhypercube it implies that expansion is required. The system eliminates duplicates and looks ahead via the Goal 'getallsubhypercubes' which returns the new stack list ready for the next level.

4.2.3.10 Getallsubhcubes.

The predicate 'getallsubhcubes' has the format: **getallsubhcubes***(Top,Newstack)*. The purpose of getallsubhcubes is to eliminate duplicates and perform a look ahead function thereby again eliminating unnecessary processing at the next level. The necessary elimination of duplicates, accumulated from the firstbeam search, again points to the need for a double beam algorithm; the alternative single beam search being very wasteful of resources.

Getallsubhcubes builds the list of subhypercubes for the given subhypercube 'Top' as two lists. 'Onelist' is Top plus all possible combinations of ϕ-boolean variables converted to '1' and reduced if a duplicate or if it contains no new information. 'Zerolist' is the analogous case for '0'. Onelist and Zerolist are concatenated to form 'Newstack'.

```
getallsubhcubes(Top,Newstack) :-                % plus 'stackcheck' for duplicates
        buildsublist(Top,[],[],Onelist,[],Zerolist),
        conc(Onelist,Zerolist,Newstack),
        addtosubstack(Newstack).                % add to substack
```

The 'addtosubstack' clauses assert the Newstack list in a 'substack' dynamic database for later comparison as a duplicates test. In 'addtosubstack' a unary relation fact forms the null case, the first rule body is a single 'Goal' and the second rule body is a dual 'Goal' recursive multiple list test.

4.2.3.11 Buildsublist.

The predicate 'buildsublist' has the format:
buildsublist*(Top,Front,Onelist,Onelist2,Zerolist,Zerolist2)*.

```
buildsublist([],_,L,L,L1,L1).
buildsublist([X|R],Front,Onelist,Onelist2,Zerolist,Zerolist2) :-    % unknown
        var(X),
        makeconclist(Front,1,R,Ones),
        makeconclist(Front,0,R,Zeros),
        trysubstack(Ones,Onelist,Onelist1),
        trysubstack(Zeros,Zerolist,Zerolist1),
        conc(Front,[X],Front1),
        buildsublist(R,Front1,Onelist1,Onelist2,Zerolist1,Zerolist2).
buildsublist([X|R],Front,Onelist,Onelist1,Zerolist,Zerolist1) :-    % 1 or 0
```

```
conc(Front,[X],Front1),
buildsublist(R,Front1,Onelist,Onelist1,Zerolist,Zerolist1).
```

The first clause exits the recursion with Onelist and Zerolist. In the first rule head X is the ϕ-boolean case and therefore builds the new subhypercube list with 1's replacements - see above comments with respect to reduced in section 4.2.1.10. Similarly for 0's. The predicate 'trysubstack' then considers duplication. A number of questions can be asked of this and ensuing parts of the algorithm; two of which we will briefly mention.

Firstly, does this imply that only unique subhypercubes are put on the stack? Secondly, to what extent is this possibly extensive code for uniqueness worth it, especially timewise? Tests showed that eliminating the two trysubstack's could considerably increase the algorithmic speed, however, there is then no test for duplicates - which is an explosive deficiency. A possible solution is 1 -> [] and [] -> 1 at the end of the program when the formula is found. This reserves 1 -> x for the 'seen' case - it then being unnecessary to invoke trysubstack. The buildsublist rules are again in need of iteration conversion to reduce garbage collection.

The arguments of the 'makeconclist' predicate are used each on each recursion of 'buildsublist' to build the new subhypercube with the variable converted to either 1 or 0 by triple concatenation. A problem is that the makeconclist procedure is exceedingly expensive. Extensive inspection of this tight loop using the 'trace' facility suggested that there is much duplication of effort. Hence we might alternatively consider for example:
L = [1, _, 0], together with its associated templates: L_1 = [_, 1, _] and L_2 = [_, 0, _].
The big advantage gained is unification! Matching then gives:
L_1 = [1, 1, 0] and L_2 = [1, 0, 0].
This was found experimentally to be 50% faster. A requirement, however, is for "index" obtained from the length of 'front' in buildsublist for 'size' of 'front' which then indexes the templates. It is also dimensionally dependent. An example format is:
template(*[Size, [_, 1, _], [_, 0, _], D).*

The disadvantage is that all possible permutations of templates are required. The solution is therefore **ugly but fast**. Worse still, combinatorially it clearly will not fit the general case where the present solution is the default. (Later refinements very elegantly eliminated these problems and moreover further improved in both space and time the advantages).

4.2.3.12 Trysubstack.

The 'Goal' 'trysubstack' checks for duplicates via 'insubstack' and looks ahead to see if the binary relation 'h' exits via 'testcase112'. The subclause procedures at this juncture

are fairly complicated and tricky. Their implication opaqueness, especially at runtime, is partly caused by their depth in the recursive tree. (The rule of seven problem again). We therefore "gloss over" and summarise as follows.

Utilising the logical boolean truth 'Reply' returned from 'insubstack' and the logical boolean truth 'Ans' returned from 'testcase112' the predicate 'maybeadd' either does or does not append the list to the current 'Onelist' or 'Zerolist' as the second argument of 'trysubstack' in 'buildsublist' in order to produce 'Newlists' which is the new 'Onelist1' or 'Zerolist1' argument accordingly in 'trysubstack' in 'buildsublist'.

```
trysubstack(List,Oldlists,Newlists) :-
        copy_term(List,List1),
        copy_term(List,List2),
        insubstack(List,Reply),                    % check for duplicates
        testcase112(List1,Ans),                    % look ahead!
        maybeadd(List2,Reply,Ans,Oldlists,Newlists).
```

Some questions which were later experimentally addressed with respect to the above were: how does the time and computation generally for increasing dimensionality compare when computing: none, one, the other or both 'insubstack' and 'testcase112' procedures? This would also imply the possible elimination of 'addtosubstack' and its call in 'getallsubhcubes' for one or both cases when eliminating the duplicates check of 'trysubstack'.

Many of the procedures involved in the duplicates test for unique subhypercubes are very expensive. For example, to avoid ϕ-boolean masking instantiations requires an expensive remoulding ϕ-boolean procedure called 'converttodash'. E.g. unification with:

[..., _, 1, ...]

is polysemous by instantiation hence leading to multiforms; whereas specifying a 'dash form' means matching that particular form stored dynamically in the substack and is therefore exact as in:

[..., _, 1, ...] <-> [..., '_', 1, ...].

The basic problem is that it is hard to avoid the 'x', 'dash', etc. conversions being performed on every node in the subhypercube and then repeated, often on the same nodes, down through the levels of the hypercube.

One improvement is to work with "types" (e.g. 1's, 0's, ϕ, etc.) testing all of that type by ϕ-boolean masking, this implies stacking by type. Further, there is no point in testing the substack if in fact the subhypercube does not exist, especially so since its subclause the 'ensuresame' procedure is expensive. This implies invoking the 'Goal' 'trysubstack' if and only if 'testcase112' procedure's reply is false. This leads to considerable modification of 'buildsublist' and 'trysubstack' and both argument and clause changes to 'maybeadd'. We will not elaborate on this.

4.2.4 Analysis of the learn algorithm.

We have discussed the first global subhypercubing algorithm algorithm global0. We have also discussed some of the considerations, tests, etc. which were involved in the further development of this algorithm (as section 4.2.3). A listing of the final subhypercubing algorithm called "**learn**" is given in Appendix 7. This enables our analysis of learn to be brief, in note form and very high level. Public predicates within the module system may be imported and exported. All procedures in prolog fall into one of two categories: static or dynamic.

4.2.4.1 Imports and exports.

Imports: the algorithm takes its hypercube database 'h' input (e.g. parityeven15) and the equivalent specified generated 't' file (e.g. gent15) from specified database input streams. **Exports:** output goes to a specified output stream.

4.2.4.2 Dynamics and statics.

Dynamics: there are 21 databases which are required by "**learn**" (some of these are trivial) and these dynamics are set up and defined in terms of **predicate***(Arity)* as follows:
h/1, t/4, stack1/2, stack 2/2, formula/2, dimension/1, resolution/1, b_chop/1, noise/1, top_bot/1, level/1, varsposition/1, vold/1, supervars/3, s_mode/1, debug/1, vpos/1, front/1, middle/1, rear/1, listformula/1. **Statics:** are the algorithmic procedures.

4.2.4.3 The learn procedure.

The subhypercubing algorithm is invoked by the goal "**learn**" and has essentially 3 major components:

1) An initialisation procedure "init_learn."
2) The heart of the algorithm is invoked by the subgoal "search_levels."
3) There are various concluding procedures.

The **init_learn** procedure: is comprised of a large number of procedures which although interleaved can themselves be grouped into 3 major parts:

a) A precautionary programming database pre-initialisation procedure called "undo" for the 21 dynamics, which fails safe via "retractall". (Recall the point in section 4.2.3.5.4).
b) A set of initialisation procedures to build the initial state of the dynamic databases.
c) A record of the particular set of invoked parameters, files, inputs, etc. is then sent to the output stream.

To a considerable extent the power and speed of the algorithm results from the care and attention to detail with which this defined algorithmic environment is created. We will briefly highlight a few of the main points.

1. Input streams are checked for type and range of input, leading to several possible default and exception clauses in each case.

2. A new faster incrementing "incmember" variable position "Vposition" list constructor refinement is exponentially advantageous with respect to the prior constructive, concatenation-based "build_v" structure. Vposition is crucially important to the marvellous speed gains of "supervars" particularly when avoiding recursion (see later). Examples of these gains are given in Table 4.6.

D	N	I	G = N/I
dimension	nonincremental goals	incremental goals	gain
3	28	13	0.50
13	163	43	0.25

Table 4.6 Incrementing variable position list constructor gains.

3) Unlike most machine learning and connectionist algorithms where the learning time cannot be calculated prior to learning, the system is able to output estimates as guide-lines to the user on the maximum probable learning time by taking into account the particular parameter set used in the problem. Further work in this area should even enable a reasonable estimate of the learning time for *any* problem. This is possible by comparing the parity distance of exemplars via their percentage linearity/non-linearity calculus.

4) Advice based upon the parameter set interactively input so far guides the input settings of the remaining parameters.

5) The algorithm automatically caters for parameter coupling as in 'top_bot' and 'b_chop' (see later notes).

6) The procedure "check_database" ensures correct database instantiation as a prerequisite for learning and otherwise sends a warning to the output stream - solving the problem in section 4.2.3.3.4. However since the database is assumed to have been set up correctly as associative, there is no check for non-associativity.

7) The four trivial hypercube instantiations: I, O, 'variable-based' and 'missing database' are handled appropriately.

8) The various dynamic databases are also constructed.

Further points are that the init_learn procedures inspect the databases and act accordingly - for example by building the initial stacks etc. in the correct form. Given the

dimensionality there are several possible program variables such as the resolution and noise settings levels, also boolean choices concerning the search dynamics and search constraints such as binary chop and 'top_bot' (see later notes). It should be noted that init_learn itself is approximately a time constant - the number of 'goals' being independent of D and approximately 200 in number.

4.2.4.4 The search_levels procedure.

We have presented the basic ideas (and their variations) behind the global algorithms in section 4.2.3. Learn itself could be similarly dissected but space restricts us to discussing solely some of the more interesting additional aspects. For maximum speed recursion is shunned in the goals and subgoals of all crucially expensive code in favour of the more expedient **iterative** format "repeat ...!, fail" which relies upon throwing away the clause upon reaching a 'fail' then backtracking to the preceding 'repeat'. The solution to avoiding duplicate subhypercubes being stored is a procedure called 'makesupervars' which stands for make super or meta variables.

"**Depth resolution**," R, sets a limit on the search depth and is used when the complete solution is not required. As R tends to the parity distance of the universe the formulae tend to 100% accuracy. The usefulness of depth resolution is, as one might expect, problem dependent. In favourable circumstances (large parity distance i.e. tending to linearity) and when the dimensionality is high, then first level resolution will provide approximately 2*D division of the problem (compare section 4.2.2.2). For example, if D = 50 then we attain 1/100 = 0.01 error or 99% accuracy - without the need to search further. This is significant.

Noise represents the estimate of the amount of noise in the universe of exemplars. The noise parameter may be set at a value from 1 (no noise) to 4 (very noisy data). By increasing the noise value the algorithm tolerates higher amounts of noise in the subhypercube. It operates on an analogous "**shrink and dilate principle**" to mask out the noisy data (see later).

Top_bot is a boolean where true => we wish to utilise the Top_bot search control facilities and false => that we do not. The procedure "get_top_bot" utilises variable-relational top and base converters so that if Top_bot is false, then the init_learn assertions of Top and Base (both true) will be used throughout, otherwise Top and Base will be altered for each new expansion accordingly. If Top_bot is set to true and the search for the top or bottom of the hypercube is successful in detecting a '0' then the size of the subsequently generated subhypercubes is halved. (In practise it is more complex than this). Clearly after a few levels this has a *very marked effect on the speed* of the algorithm *at no loss of accuracy*.

Binary chop: considers *all* nodes in the universe, but it ignores all "bridging subhypercubes" between the major opposing pair of subhypercubes which is being considered at each level. An example is [_,1, ...] and [_,0,...]. Thus it has a route to all base nodes and performs a binary subhypercubing chop, hence "b_chop," at each level. The saving in time is enormous at the possible expense of accuracy. Interestingly, the accuracy is proportional to the *non-linearity*. This is a very useful side-effect since, for example, the complete solution is still found for the most non-linear of problems, i.e. parity; whereas highly linear problems are tractable anyway, for example by Top_bot. This leaves intermediate complexity problems to be most easily tackled by Resolution. It would be useful further work to see if pre-inspection of the exemplars (by estimating the complexity) would allow the automatic choice of the best strategy.

4.2.4.5 The concluding procedures.

These procedures also print the resulting formulae which have been learnt to the specified output stream and tidy up the dynamic databases, etc. The format of the formulae is: **formula***(Indicator, [Class / Subhypercube])*, where the 'Indicator' takes the value 0, 1 or 2 in order to differentiate between a classification of true, false or incompletely processed subhypercubes (as resulting from resolution) respectively.

4.2.4.6 The major experimental results.

Some experimental results of interest using from the development of "learn" are:

1 Parity in the 10th dimension by "binary chop" takes learn only 73 seconds. This is very fast. Although due to the exponential explosion in complexity higher dimensions are soon taking appreciable times.

2 Making the procedure makesupervars (see later) failure driven improves the time from 1 day to a mere 95 seconds! This remarkably positive result is at least partly due to the way QUINTUS PROLOG works (because of the elimination/reduction in garbage collection) regardless of the advantages of the algorithmic subtleties of iterative prolog. At lower dimensions the effect is much less marked.

3 Rewriting the 3 concatenation makesupervars as single concatenation machine speeds up the algorithm by only 1 second at the 10th dimension. An equally remarkably disappointing negative result - well within experimental error - which is a pity since it was a nice idea!

4 Making h(I,C) as unified list as in h([C|I]) for faster indexing, referencing, unification, etc. speeds up the algorithm from 95 to 73 seconds at the 10th dimension.

Other changes had less dramatic effects. Often the advances are difficult to illustrate graphically and the gains only minor but cumulative. In contrast, the elimination of recursion from global0.8 to global0.9 (learn) had sufficient effect on efficiency to be clearly seen even in the logarithmic graph of figure 4.34 where global0.9 is far flatter. Since the scale is logarithmic the noted speed-up is very considerable where gains are hardest won, that is, at the higher dimensions.

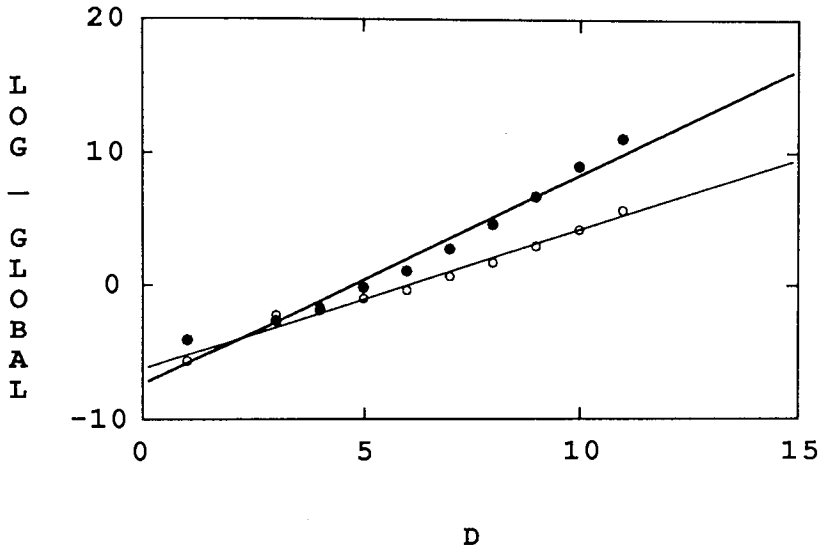


Figure 4.34 Comparing recursion and iteration.

4.3 TESTING THE PREDICTIONS OF THE THEORY.

As we stated in section 3.1 the **purpose** of experiment lies in "**testing the predictions of the theory**". This we did in accordance with our scientific cycle outlined in section 3.1.3. Yet experiments cannot, of course, prove the theory (section 3.1.2). We merely, at best, retain the theory pending more exacting experimental demands. Our conclusions regarding the testing of the theoretical predictions may therefore prove to be ephemeral.

4.3.1 On the experimental wrinkles.

We ought first to say a few words about the experimental approach that we have taken throughout. The object of all these experiments is, as far as time allows, to identify the general and particular *behaviour* of the machine. Thus we are concerned to find the limits and boundaries of the learning machine, what it generally will and will not do, whether it enacts

the theory or not, and to discover *experimentally* what it is designed to do and not to do. This includes some feel for the computational resources, CPU and memory requirements of the system. (These later points we have already reported in some detail in our account of the development of the machine, especially with regard to the parity experiments, see section 4.2).

In experimenting have taken many precautions. For example, we have been very concerned to run (multiply) repeated experiments (as in section 4.1.1.3). There are always new tests to perform with possibly a nasty or unexplained surprise in store. For example, some new unsuspected source of inefficiency was occasionally uncovered (e.g. section 4.2.3.3.2). Other precautions have been: to ensure convergence in reasonable times by mini trial runs, varying everything that can reasonably be varied, inserting deliberate spy points, etc.

We have attempted to take on the characteristics of the investigator by looking *afresh* at the subject with the conviction that since nobody knows what the subhypercubing machine can do, the answer is *only* to be found by experiment. Role change is required here. Such investigation without preconceptions has involved convincingly "exchanging hats" and putting aside previous roles of theorist and developer of the machine to examine the machine *afresh.* This time experimentally.

We have been at pains to be very awake to the possibility of difficulties, error and misinterpretation. For example, one graph (Figure 4.15) had a highly significant curve (we were expecting a straight line and got a straight line, if somewhat wobbly), fortunately a healthy scepticism about the wobble led to rechecking everything about the experiments. (Counterintuitively, we have discovered that such is the way towards *confidence in the results*). One of the points had been incorrectly transcribed. The new graph looked very different. Such a painstaking approach paid dividends several times, particularly with regard to rechecking experiments after modifications to the algorithm and in the use of trace facilities. We have also attempted alternative calculations, graphs, etc. where possible - again as a check, as with for example Figures 20 and 21.

The melting pot of indiscriminate averaging needed to be intelligently examined - *just in case* . . . A case in point, for example, is our comments with regard to the first two points of Figure 4.19, where the variability was "overridden" i.e. the data was *not* discarded in any respect, its significance having been spotted. A reverse example is the point about the first two readings of a run being consistently very much higher (section 4.2.3.3). In this case the data *is discarded.*

The almost impossible difficulties with Pollux and Castor encouraged a calm expectation for the details to vary grotesquely during the experiment, so that scepticism and

cross-checking became second nature! And so it was after all good training. We conclude that a poor environment wastes valuable time, but encourages good experimental techniques.

There are a number of spin-offs. First, all this has also led to an appreciation of a detached appraisal of the general reasonableness of the results. Secondly, we have gained some "feel" for the machine, some knowledge of its behaviour, an expectation at least by order of magnitude and often better. Having approached our experiments on the machine so *experimentally* we now have some confidence in our verification of the correspondence between the developed machine and the underlying theoretical ideas of chapter 3, as shall now see.

4.3.2 Testing the theoretical requirements.

In section 3.4.4.3 we summarised the theoretical algorithmic requirements as the following quick reference word list:

1) simplest
2) global
3) balanced
4) correspondence
5) targeted
6) faster
7) virtual
8) actual
9) ϕ-boolean
10) associative
11) resolution.

The first three requirements express a strong preference for the **simplest** possible **globally** derived explanation that fits the data in a **balanced** way with respect to both positive and negative evidence (section 3.3.3, 3.4.4.3 and 3.4.5.3). If the reader will recall these preferences arose out of our experience with the psychology tests. As a result of that study our **reasonable theory** had to give way to our later **unreasonable theory**. It is worth while briefly repeating the essentials. In section 3.3.3 we speculated that:

"A machine needs to be built as an implementation of this extended theory. Experiments are required to test the performance of the machine. Clearly, all this is going to occupy the rest of this project. We postulate the machine equivalent of the unreasonable theory, the **MEU theory**.

If there are no external constraints, the generalisation process will not choose randomly among abstract possibilities; but rather, will always choose the simplest possible globally derived generalisation".

We also concluded in section 3.3.3 that the learning machine should have a strong preference for :

"the simplest possible globally derived explanation that fits the data in a balanced way with respect to both positive and negative evidence."

Having built our implementation of the **MEU theory** in accordance with the "strong preference", we are now in a position to reconsider the matter. We required our machine to operate in accordance with our psychological experiment's profile of ourselves for all the reasons previously given in section 3.3.3 - e.g. in preferring simplicity. Essentially, *we have deliberately sought to define an* **anthropomorphic machine** in this regard.

There is a beautiful way of testing whether our machine reaches the same conclusions as we do - bearing in mind the very remote *random* chance of this happening (as section 3.3.3). The answer is to simply present the same psychological tests to the machine. This set of tests is uniquely well selected. It provides the precisely focussed "point of possible failure in the theory" that we demanded of a well designed experiment in section 3.1.3. The result of these tests is that the machine does indeed reproduce our psychological test result solutions correctly. Had this not been the case we could have *disproved* the theory (section 3.1.3). So the theory is retained for the present. We must **not** fall into the tempting trap of saying: "This result automatically satisfies our three criteria". (A notable side observation is that: the machine solves these problems several orders of magnitude faster than human subjects).

The fourth and fifth requirements were for a 1 : 1 **correspondence** with cr for all possible functions in the universe Θ, so that, by correspondence with cr, the algorithm is mathematically analysable in depth, yet *does* refer directly to the **target** global function(s) of the given problem (as sections 3.4.4.2, 3.4.4.3 and 3.4.5.3). Ideally this would require an infinity of tests for correspondence. (Subhypercubing is always on target). Further, we can rarely consider all "possible functions" and then only for the simplest of universes. Again we look for tests with considerable purchase at a point of possible failure in the theory.

In section 3.4.5.3 we illustrated the simple case of H^2 with respect to **number and ordering of the function types**. Our implemented learning machine can be experimented with to see if it *fails* to follow our "**convention for the search order**" (section 3.4.5.3). In fact the earliest of our implemented subhypercubing algorithmic versions already showed this 1:1 correspondence *within its complexity range criteria.* In further detail, for the algorithm global0.2 the function types for the first case of D = 1 are given in Figure 4.35 where critical range is plotted against the time taken. The two cr = 1.5 cases were each averaged times corresponding to even and odd parity. The remaining points at cr = 0.5 correspond to I and O. Although D = 1 is fairly trivial, we can still see a distinct separation between function types. This should **not** be so if we are to refute correspondence or

targetting. The 5 function types for the second case of D = 2 are given in Figure 4.36. The 5 averaged results on the graph may be compared in their ordering with the theoretical prediction (Figure 3.2.4). Again the correspondence fails to be refuted. As seen in Table 3.6 (fourth table) the functions rapidly explode after D = 2 reaching 218 by D = 3. Nevertheless if we examine other relevant experimental results for these higher dimensions we have still failed, so far, to refute the 1:1 correspondence and targetting requirements of the theory. Once again the theory is, for the present, retained.

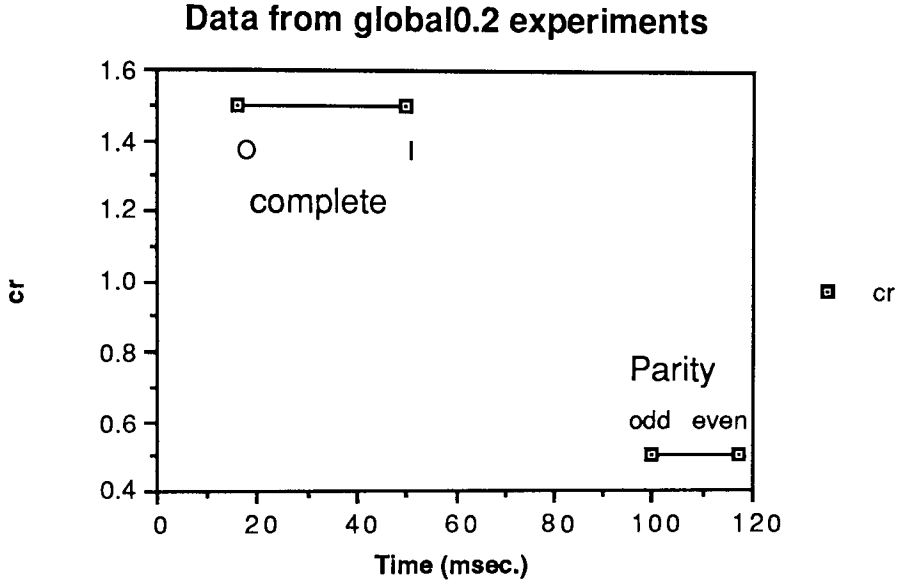


Figure 4.35 Function types for D = 1.

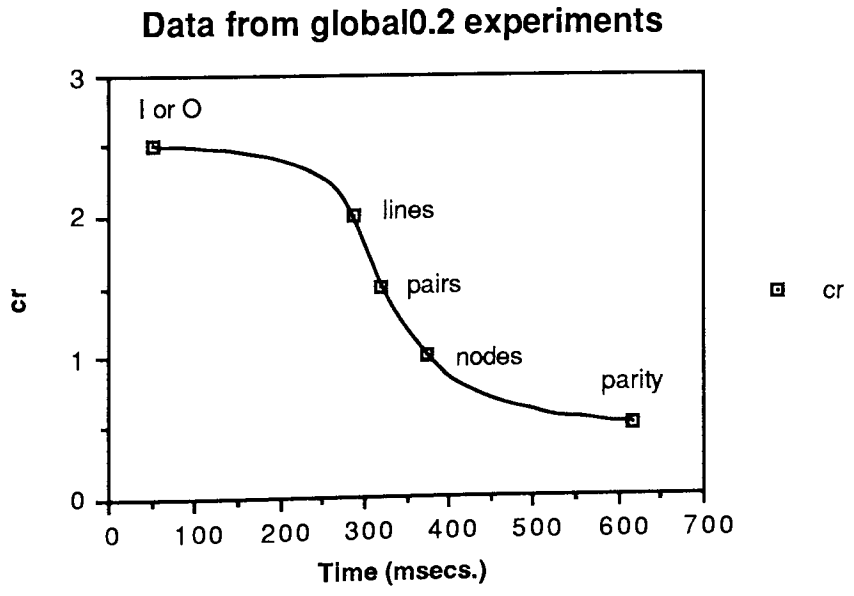


Figure 4.36 Function types for D = 2.

The sixth requirement was for a *very* much **faster** algorithm to fight the NP problem (for as long as possible) by considering *groups* of nodes (comprising perhaps millions of nodes at a time) *in parallel*, rather than simply considering individual nodes one at a time as in cr (section 3.4.4.2). This is, of course, how the implemented subhypercubing algorithm works, approaching the solution from both the most specific and most general possible concepts within the hypercube lattice (section 4.2).

It is important to notice that learning can partly be seen as an exercise in handling complexity most efficiently in order to fight the inevitable explosion. Only two paradigms have so far been used:

1 The use of heuristics (in Machine Learning and Knowledge Acquisition).
2 The use of sparsity (in Connectionism, Genetics and Statistical methods).

In the both cases, wholesale elimination of large parts of the space reduces the size of the problem. Typically the algorithms employed are, as we have seen (section 3.4.5.5), "local". But whether these algorithms are local or global in operation the "elimination" is at least in some measure *random.*

We contrast this with our **global** and **orthogonal** algorithm. That is, even though it is not possible to consider all possibilities, yet that was precisely our starting point. Our algorithmic versions then steadily accumulated methods by which sparsity may be usefully employed. The wholesale elimination is then done, *through the orthogonality*, or as we have called it "**metrically**" (section 3.4.2, 3.4.4, 3.4.5, 3.5.7 and 3.5.7.1).

We have in mind here the idea that the metric allows us to know where we are in the space by reference to a metric related to the universe, namely a given subhypercube or in the general case a polytope. Here is the crucially important advantage over other methods as seen for example in section 3.2.5. In this regard we have a **novel third paradigm** as an approach to the speed and complexity problem. For this reason, it is possible that this concept could in time prove to be an important contribution of the thesis.

The theory further required:

1 **virtual** storage of the non-exemplars as a connectionist sparsity within the hypercube in order to avoid memory overflow in the general case (section 3.4.4.2 and 3.4.4.3).

2 **Actual** storage of exemplars and their associated classification in order that the algorithm may learn (generalise) from these exemplars (section 3.4.4.1 and 3.4.4.3).

3 **ϕ-booleans** in order that the algorithm should cope with groups of nodes (generalisation) as well as individual nodes (specialisation) as section 3.4.4.3.

4 An **associative** hypercube for its advantages (section 3.4.2 and 3.4.4.3).

5 A flexible means of adjusting the depth of search or **"resolution"** according to circumstances in order to speed up the algorithm and make accessible the solution of very large problems (section 3.4.4.2 and 3.4.4.3).

As we saw in section 4.2 this is precisely how the development machine operates. Hence we must conclude that: "we have been **unable** to demonstrate experimentally that the machine does **not** act in accordance with the predictions from the theory". Strictly speaking we cannot reverse this statement and tentatively conclude that: "*for the tests so far undertaken the practical machine does appear to consistently operate in accord with the theoretical predictions*". Although the temptation is to do just that.

4.4 SCOPE AND LIMITATIONS OF THE MACHINE.

In section 4.1 we stated that: a third consequence for the subhypercube theory is that we must "fight for its survival". In building upon accepted theory the researcher can liberally quote the equations developed and checked by others and their practical validity and applicability is not immediately suspect. For the subhypercube theory we have a second major exercise demonstrating the practical applicability of the theory by addressing the scope and limitations of the machine. That we now consider.

4.4.1 Scope.

In addressing the practical work we had in mind the need to involve certain basic criteria. Firstly, to scale up to about the fifteen dimension for complete convergence even in the face of parity. That we achieved as already discussed. Approximations were then introduced for higher dimensions. This we discuss shortly. The objective was to determine the "best" algorithm for the parity problem which was then considered to be our best algorithm.

Next we needed to apply this "best" algorithm to appreciably different problems with the objective to determine the scope of the approach. We needed to find out the behaviour of the machine in different application areas. Performance issues also had to be considered for appropriate problems in order to determine the performance of the system to noise and prediction. Clearly, time limits just how much can be achieved and space limits too detailed a description of what work has been done. However we now give a representative sample of our work in this area.

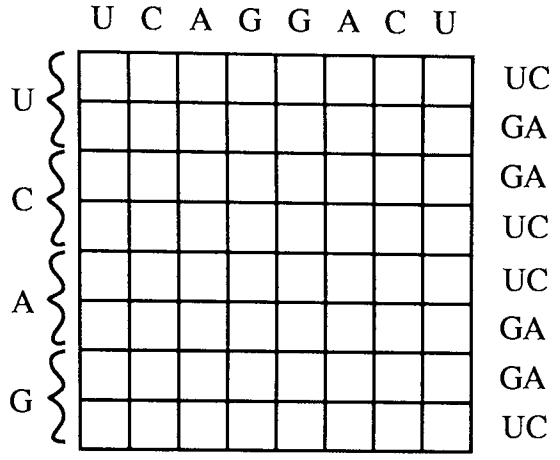
4.4.1.1 Complete convergence: mRNA Codons.

As an example of the scope of the machine using the complete convergence facility of our algorithm we experimented with mRNA codons. First a little genetics. The sequence of bases in a length of DNA determines a complementary sequence of bases in mRNA, which then determines a sequence of amino acids. There are only four bases in DNA namely: A, C, G and T. A coding group of two bases would give sixteen pairs which is insufficient for the twenty amino acids. Three bases gives 64 different possibilities; four gives 256 etc. Biological experiments support the hypothesis that the group size is three bases per amino acid. The amino acids have abbreviated names such as: ala for alanine, try for tryptophan, met for methionine, etc. Groups of amino acids form the genetic code. A codon only codes for one amino acid. But a given amino acid can be coded for by related codons (which we shall call a subgroup) since there are 64 - 20 = 44 spare slots.

We used our hypercube machine in the following way. Since biology suggests 64 possibilities is sufficient we can use a "fixed" representation. Thus H^6 rather than H^{64} is sufficient. The four bases can be coded by 00 to 11 in booleans and the three successive bases by three such groups in < attribute | value > form where the attributes are the base order 1, 2 or 3 and the value is the actual base/attribute. For example: ala is formed by GCU, GCC GCA or GCG. In the hypercube we represent U as 00, C as 01, G as 10 and A as 11. In binary GCU becomes 100100. Hence for the first ala codon in the subgroup the database format is: h([1,1,0,0,0,1,0,0]).

One or two practical points. The hypercube database was set up in "nonlearning" mode by which we mean learning in the presence of a completely specified universe. (That is U = T = P or U = T, P where U, T, P means universe, training set and prediction sets respectively). The problems were "hard" (small cr so near parity functions). Processing times were just seconds or less than a second.

Figure 4.37 (a) shows a two dimensional representation of the mRNA codons where the triple bases map as follows. The right hand side designates the first base, the left hand side each half row and the columns at the top form the third base. Since the corresponding hypercube is quite complex we give the reflective Gray equivalent coding in Figure 4.37 (b), where, with wraparound, all adjacent cells are only only bit different. The ala codon GGU equivalent of our hypercube database format: h([1,1,0,0,1,0,0]) is seen as 100100 in the bottom right hand corner of Figure 4.37 (b). As a representative example of the use of the hypercube we consider special codons which act to punctuate sequences of the genetic code. We shall call certain of these, which act as end markers to a sequence, "stop" codons. One interesting technique is to reverse the logic and find the structure of these stop codons. The stop codons are UAA, UAG and UGA. Notice that C does not occur at all in these codons.



(a) Triple Codon structure defined as code below.

000000	000001	000011	000010	000110	000111	000101	000100
001000	001001	001011	001010	001110	001111	001101	001100
011000	011001	011011	011010	011110	011111	011101	011100
010000	010001	010011	010010	010110	010111	010101	010100
110000	110001	110011	110010	110110	110111	110101	110100
111000	111001	111011	111010	111110	111111	111101	111100
101000	101001	101011	101010	101110	101111	101101	101100
100000	100001	100011	100010	100110	100111	100101	100100

(b) Reflective Gray coding on fixed: 8*8 grid

Figure 4.37 A coding scheme for coding mRNA codons.

Formulae for C are [0,1,_,_,_,_], [_,_,0,1,_,_] and [_,_,_,_0,1]. This covers 37 codons as illustrated in Figure 4.38 (a). [1] We suggest the hypothesis that, in effect, this makes the codons "safe" for reading purposes within a sequence. These and their related codons can therefore be predicted to form the largest of such subgroups; which is indeed the case (typically four in size). Each stop codon begins 0,0,... This rules out 01 (C), 10 (G) and 11 (A). Each specified binary in the connection forms half the hypercube. Two specified binaries therefore occupy 1/4 of the space. The first base therefore rules out 3/4 of the space of codons as Figure 4.38 (b). The second base for stops' must be G or A. A similar argument reduces the space of possible codons to just 8 as illustrated in Figure 4.38 (c). Repeating the exercise for the third codon leaves just four codons as Figure 4.38 (d). The three stops are all found within this very much reduced space.

The relationships of the stop codon sequences are illustrated in Figure 4.38 (e). If the genetic code were to have some degree of protection against reading failure then we would expect the outstanding fourth codon (UGG "try" dotted in Figure 4.38 (e)) to be a punctuation sequence. This is indeed the case. Further, practical linkage with this sequence should include any outstanding punctuation sequences. Again this is the case. There is only one: AUG (dotted in Figure 4.38 (e)). AUG ("met") in fact marks the starting point for a polypeptide sequence and is again ensured fail-safe to some extent by eliminating the C base. Thus the smallest groups are all punctuation sequences and are all linked and heavily reduce the possibility of reading errors as Figure 4.39 where checkerboard marks the stops and black the remaining punctuation codons. Again, learning other subgroups reveals two further tricks: repeated bases (in two codon subgroups e.g. UUC) and base independence within the codon (two bases are common to all codons in a given amino acid).

The hypercube reveals the structure of the mRNA codons to be almost intelligently designed, as very compact and a good instructive source for tips on designing compact safe codes. [2] Doubtless biologists know all this but the hypercube's ability to find the exact solution with ease provides a useful tool for inspection purposes. In fact the reverse process could be considered: that is, using the hypercube to design useful fail-safe codes. The hypercube can be reliably used in this way up to at least H^{15} (for a learning time of less than 24 hours) and somewhat beyond in many cases. [3]

---

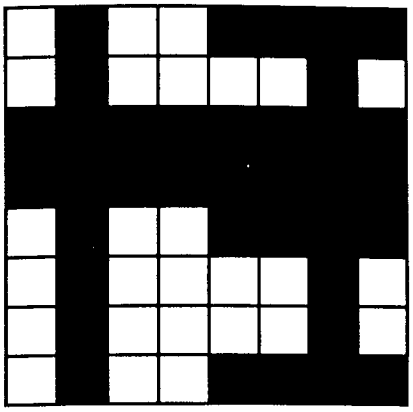
1 We calculate the general cover formula for this type of problem, where there are shifted groups of m bits along a string of length m+n, for m fixed bits and n variable bits, to be:

$$2^n (1 + (1 - 1/2^m) + (1 - 1/2^m)^2 + ... + (1 - 1/2^m)^{n/m})$$

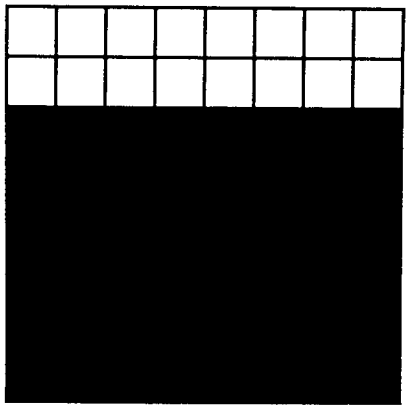
In this case, the cover C for "C" codons given m=2, n=4 is $C = 16 (1 + 3/4 + (3/4)^2) = 16 + 12 + 9 = 37.$

2 This code reliability has allowed the genetic code to become universal such that (as experimentally found by Biologists) yeast, E coli bacteria, guinea pigs and man etc. all use the same code.

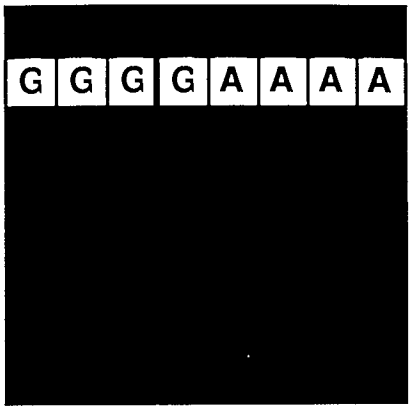
3 The human genome experiment researchers are reportedly using neural nets. Yet to illustrate the difference in power on this type of problem back propagation would already, we believe, have found the above sequence to be sufficiently difficult to have "got lost" at least some of the time in attempting total convergence in the sixth dimension.



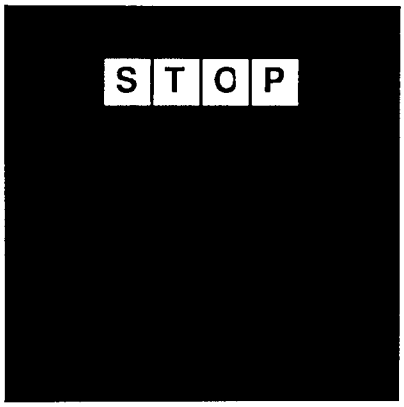
(a) Eliminating C



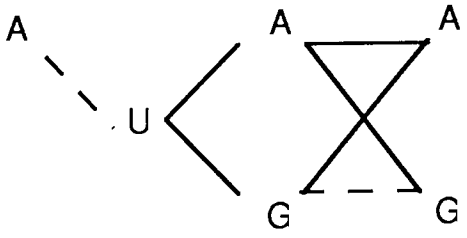
(b) First Base is U



(c) Second base is A or G



(d) Third base is A or G



(e) Triple Codon stop structure

Figure 4.38 Finding the stop structure in mRNA codons.

Thus similar but much more complex problems in genetics (e.g. polypeptides, ribosomes and possibly simple proteins) and other domains could be usefully tackled.

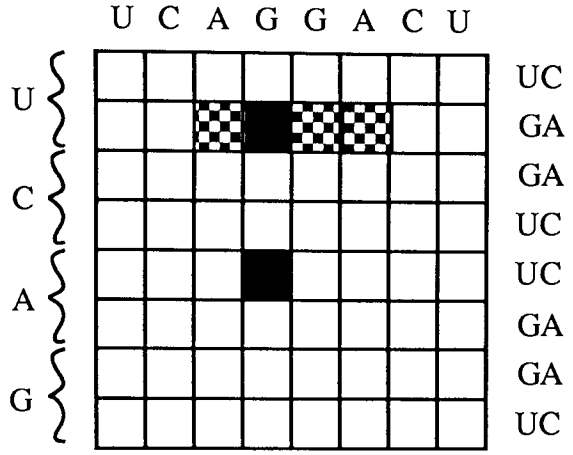


Figure 4.39 The punctuation codons.

4.4.1.2 Comparing with Boole and C4.

We were lucky enough to find two papers where we could compare our machine performance on the same experiments with the figures given in the papers. The machines concerned were a well known genetic learning system called "BOOLE" by Wilson (1987) and an ID3-like decision tree classifier called "C4" by Quinlan (1988).

Unusually, sufficient experimental details were given to allow such a comparison e.g. same experiment, same data, same hardware (sun 3/50). The learning task concerned is in the multiplexer family where a given string is fronted by n address bits and followed by the corresponding 2^n data bits. The classification of the string is positive if the equivalent data bit indicated by the address bits is on and negative otherwise. Hence positive classification for a string described in terms of the six binary-valued attributes: x_0 to x_5 (i.e. F_6) satisfies the expression:

$$x_0 = 0, x_1 = 0, x_2 = 1 \quad V \quad x_0 = 0, x_1 = 1, x_3 = 1 \quad V$$
$$x_0 = 1, x_1 = 0, x_4 = 1 \quad V \quad x_0 = 1, x_1 = 1, x_5 = 1$$

which is disjunctive with mutually exclusive terms and hence "hard". Further details and various subtleties contained within the task are to be found in the papers. Briefly our comparative results are as follows:

Algorithm	trials	Training set size	Time for approx. 90% accuracy	Time (secs) for 100% accuracy
BOOLE	3200		36++	
	12000			140++
C4		100	37 to 74	
		200		41 to 78
Subhcube		8	0.5	
		12		1.5

Table 4.7 Multiplexer comparisons for F_6.

Algorithm	trials	Training set size	Time for approx.90%	Time (secs) for100%
BOOLE	70000		2880++	
	?		?	?
C4		700	758+	
		?	?	?
Subhcube		16	16	
		24		384

F_{11}

Table 4.8 Multiplexer comparisons for F_{20}.

We conclude the following major points. For this type of problem the subhypercube and BOOLE scale patterns up slower than C4. C4 and BOOLE scale up in time slower than the subhypercube. The subhypercube is significantly faster on smaller tasks and can more easily reach complete convergence. The best algorithm would depend upon the problem size with preferential ordering from lower dimensions to higher: Subhypercube, C4, BOOLE.

4.4.1.3 Finite state machines.

Finite state machines are an old technique but they have considerable power. The hypercube seems to be quite well suited to this kind of task. Consider a very simple example as Figure 4.40.

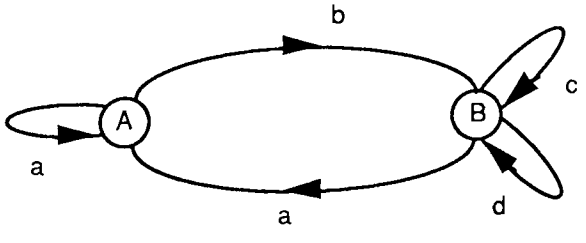


Figure 4.40 Finite state diagram.

Given some machine which can be in states A or B but not both and given tokens a, b, c and d as input then the next state is defined by the particular token received. The diagram can be translated into Table 4.9 where the entries in the cells define the next state. Cells which are invalid within the machine are shaded in the table.

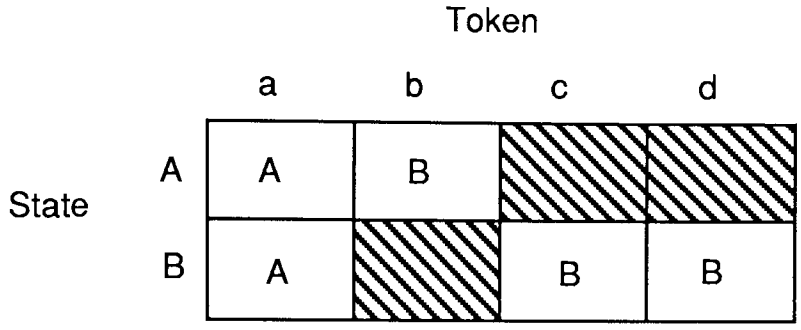


Table 4.9 The FSM state token table.

The above table can be translated into the binary classification Table 4.10. In the binary state table the first two columns labelled S define the present state and the next state. The two columns labelled T define the four possible binary tokens: a, b, c, d. The last column gives the classification, that is valid or invalid designated by 1 or 0 respectively.

S	T	C
0	0	0 0
0	0	0 1
0	0	1 0
0	0	1 1
0	1	0 0
0	1	0 1
0	1	1 0
0	1	1 1
1	0	0 0
1	0	0 1
1	0	1 0
1	0	1 1
1	1	0 0
1	1	0 1
1	1	1 0
1	1	1 1

Table 4.10 State token class.

Learning can now proceed. This simple example only took 1.295 seconds and resulted in [0,1,0,1], [_,0,0,0], [1,1,1,_] => class = 1. Therefore the binary state table has been compacted to just three simple formulae defining valid states of the machine. There are many uses for such high compaction especially in real time work where every millisecond or

less counts when checking for a bad incoming packet say. We have experimented with several variations and domains, for example:

1 Learning the finite state table from simple examples of natural language. There are many possible such uses in natural language. We have used the machine to learn morphological (Latin/Greek) semantics; learning sentence structure (noun phrase etc.) and valid/invalid word prediction (from learnt in/valid letter combinations).

2 Learning to predict the next token and using the machine to simulate and debug a small top level user interface message system, with messages such as: " ... OK, What can I do next for you?, ... " (For message read token). That is we have attempted to use the machine to design a simple dialogue with the user by simulating the system and user messages in the form of: a dialogue initiation message, continuation messages, clarification messages, help messages, complaint messages, etc. with user message responses as: task descriptions, queries, answers, etc. Parsing these responses can not, of course, be done by the hypercube. We have not tried it but object oriented design (OOD) implying objects, classes and messages could proceed similarly - for several reasons the hypercube approach may be *ideal* for OOD. One problem however is the need for examples of invalid states - which in some cases may not be available. In effect we would prefer to have available multiple output classes to define the next state. If each token has a prescribed boolean range of values by which we can indicate weights then we can simulate a neural network within the hypercube. Simulating a conventional serial machine is less satisfactory - it uses up lots of binary tokens fast.

4.4.1.4 Further experiments.

We have given, above, several representative examples of our work in investigating the scope of the machine. Space does not allow us to describe every detail of these experiments. Neither can we even mention every experiment. However to indicate the scope of the machine we will very briefly state other work undertaken. We have used the Spencer Brown logic of distinctions (Brown, 1969) as a basis for learning. The ability to forming distinctions is fundamental to any pattern learning machine. Two basic theorems in the logic are theorem 9 on variance, and theorem 8 on invariance. If we indicate the distinction by a "|" character then the theorems are respectively:

p r | q r | | = p | q | | r . and a | b | = .

There is no mistake in the second theorem! These theorems correlate nicely with our hypercube variation and invariance: "different thing which is same" e.g. size varies, and "same thing which is different" e.g. position varies. We have done quite a bit of work in this area both theoretically and experimentally: with fixed and variable shapes by considering

change of size, shape, translation, rotation etc. as indicated by figure 4.51. We have also used the hypercube to investigate the same ideas in a more abstract way using logical implications between opposing pairs of concepts within a network of such possibilities as: parallel, opposing, ambiguous and reciprocal implications.

We have worked with our local G.P. (Dr. Smith) in knowledge acquisition mode to build up a simple symptom disease representation in the field of urinary tract infections such as cystitis in order to utilise patient records as exemplar input together with the known outcome from biopsy etc. obtained from the doctor to provide the necessary classification.

One problem is the medical term "...may ..." has almost infinite flexibility and meaning. These types of medical problem all seem to have similar characteristics from the hypercube perspective they are almost all exclusively of the "any m out of n" is groupwise pathognomonic (i.e. sufficient to determine the outcome uniquely, independently of the particular combination) type or else properly pathognomonic. Such problems we have studied are easy for the hypercube. Combinations of diseases are more interesting as in: "taking history" leading to multiple disease diagnoses. However, the resulting hypercubes are then huge.

4.4.1.5 Comparing metrical/statistically biased machines.

It is instructive to consider the performance of our machine in a little more detail than hitherto. In a fine critical appraisal of the ID3 family of automatic induction machines Bramer (1987) raises many issues of general applicability to induction machines. We discuss one such problem. Local algorithms can only "see" their way by local comparisons. Therefore as Bramer states:

"The choice of the most appropriate attribute to split on at each stage of the basic induction algorithm is in one sense arbitrary (since any choice will still lead to a decision tree), but in practice is likely to be of crucial importance if the induced rules are to have any predictive value."

By way of an example Bramer offers the problem in Table 4.11 where four attributes of 12 students are eye colour, marital status, sex and hair length and which define a classification for students as belonging to either a rugby or netball club but not both. Figure 4.41 was then given as a reasonable way of classifying the exemplars.

Several different decision trees can be induced from the data, but if a different attribute had been chosen to split on it would have led to much more compact tree. That tree defines a simple choice based on sex leading to male => rugby, female => netball as Figure 4.42. Which tree is formed will affect the predictive power of the tree. Absence of critical cases will also affect prediction. Bramer suggests that since ID3 is "formula-based" it is

therefore an unintelligent process. He states that the *method* also does not help to avoid a poor choice of attribute to split on.

It is central to the thrust of our work that our *method* does provide help in seeing the more relevant attributes **because it is a global searching algorithm and makes metrically-based decisions.** (We will illustrate this in a moment). Guidance is therefore provided at source with a bias towards the simplest possible balanced solution. This will not however solve certain other problems Bramer raises such as the possibility of an unrepresentative training set, although the global perspective may help somewhat.

Let us work through Bramer's example and see what the hypercube produces as a solution. Firstly, to get the reader's "eye in" we repeat the solid Tesseract diagram for a four dimensional hypercube space, as Figure 4.43. Secondly we can replace Table 4.11 with a binary coding for the nodes such that attribute values of: brown, yes, male, long and class = rugby are represented by 1's and blue, no, female, short and class = netball by 0's. Figure 4.44 gives the equivalent universal microfeature binary coding distribution in the hypercube.

The global space of possibilities can now be easily illustrated. For example Figure 4.45 shows the distribution for female (the base cube), married (the outermost cube) and female and married (the base level). Figure 4.46 illustrates female (the base cube), not married (the innermost cube) and female and not married (second level). Therefore the hypercube takes account of possibilities which are defined in the space but may not have actually occurred in the given training set. Notice both of these examples fail to cover the universe of possibilities. From this it is easy to see one way ID3 may "split" inappropriately.

Next consider Figure 4.47 and 4.48 both of which do split the space without exceptions. Figure 4.47 shows the distribution for male (uppermost cube) and female (lowermost cube). Figure 4.48 depicts the dichotomy between blue eyes (right hand side cube) and brown eyes (left hand side cube). Now we are in a position to consider the actual training set as Figure 4.49 where the upper set of five (the training set contained duplicates) are all exemplars of rugby classification and the lower pair are classified as netball.

Figure 4.50 illustrates the hypercube's actual generalisation based upon a global perspective of this training data and results in male or blue eyes => rugby, female and brown eyes => netball. The algorithm has eliminated hairlength and marriage as irrelevant. Support for this generalisation is that there are only two cubes containing exemplars of only one class, namely the uppermost cube and the right hand side cube; remaining cubes contain both classes, implying their resolution is to be found at lower subhypercube levels. Notice the system induces a solution covering the entire universe of possibilities without exception with a decision surface which generalises to two cubes each separately occupying half of the universe (hence the "OR") and a single face (hence the "AND"). The solution is arguable

from the data, closely predictive of the global minimum and simple. When one further crucial training example reveals the split is solely sex-based the algorithm upon reprocessing the data drops both of the second terms (eye colour) in accord.

eyecolour	married	sex	hairlength	class
brown	yes	male	long	rugby
blue	yes	male	short	rugby
brown	yes	male	long	rugby
brown	no	female	long	netball
brown	no	female	long	netball
blue	no	male	long	rugby
brown	no	female	long	netball
brown	no	male	short	rugby
brown	yes	female	short	netball
brown	no	female	long	netball
blue	no	male	long	rugby
blue	no	male	short	rugby

Table 4.11 Attribute-value classification set for student clubs.

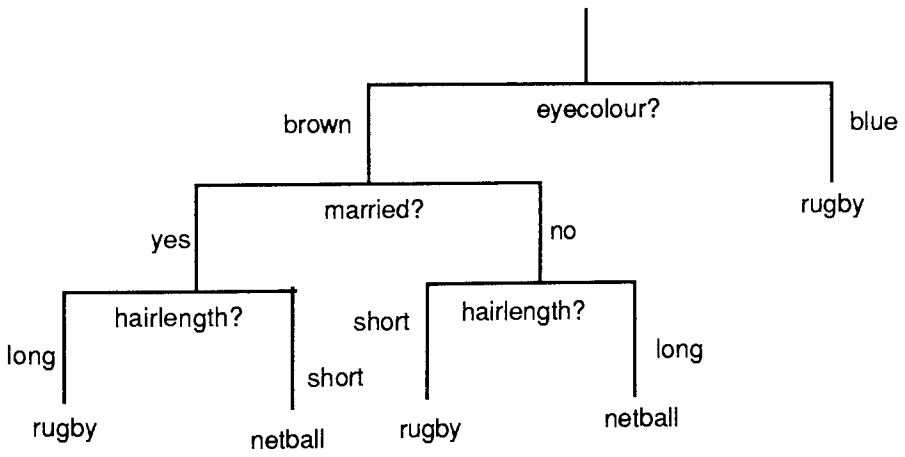


Figure 4.41 Decision tree.

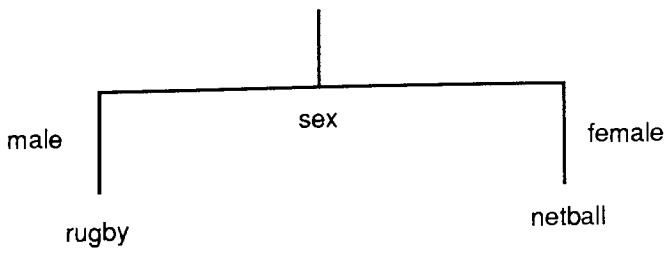


Figure 4.42 Simplified decision tree.

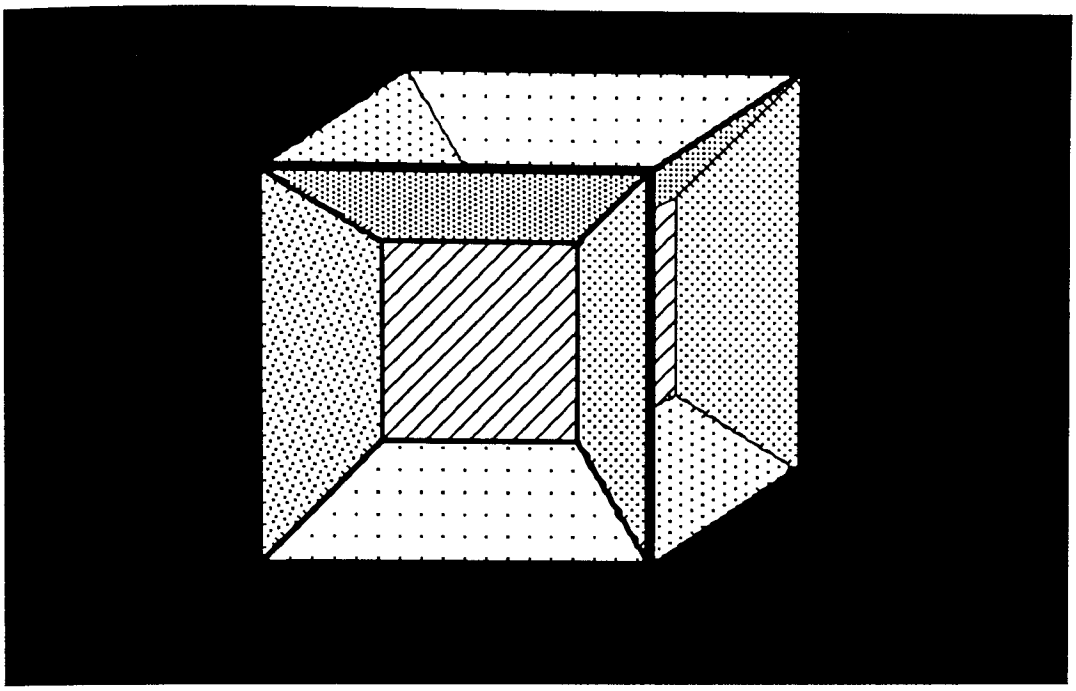


Figure 4.43 The Tesseract - a four dimensional hypercube.

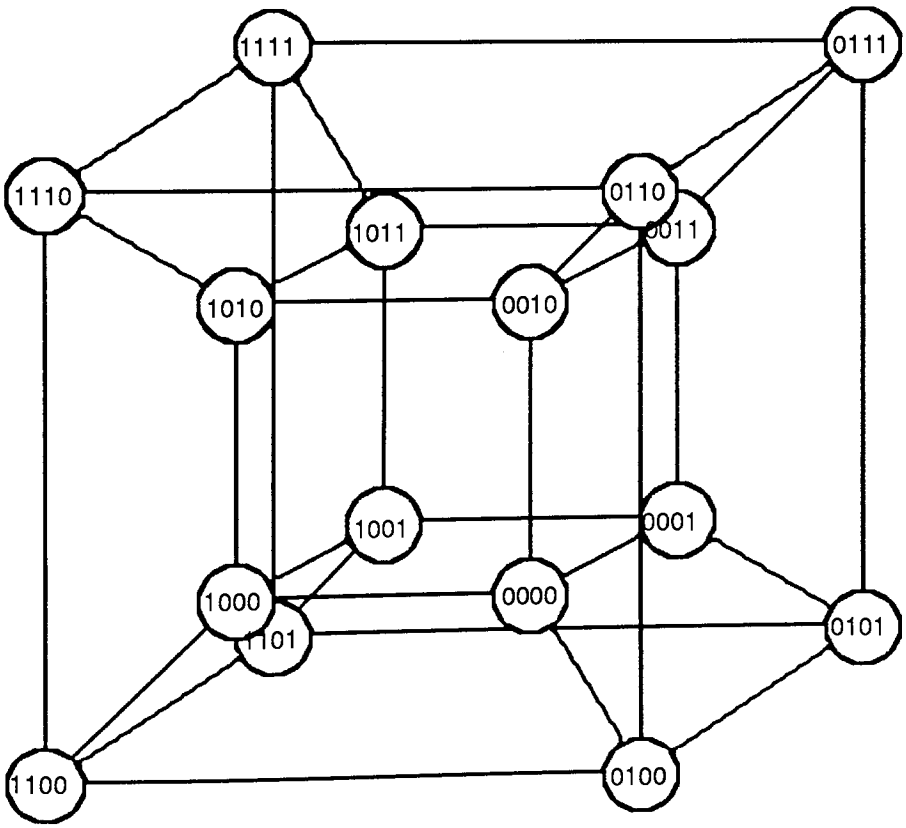


Figure 4.44 Binary labelling of the nodes within the tesseract.

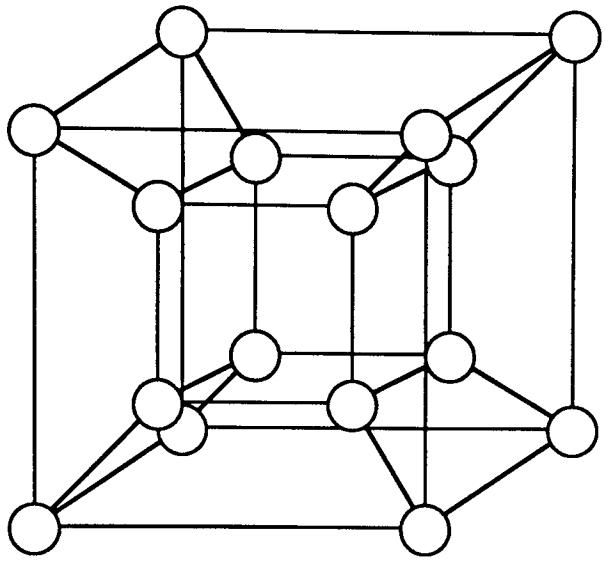


Figure 4.45 Distribution for female and married.

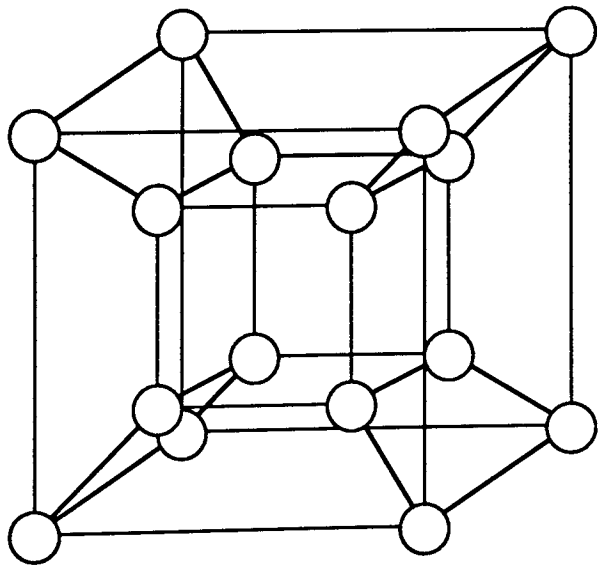


Figure 4.46 Distribution for female and not married.

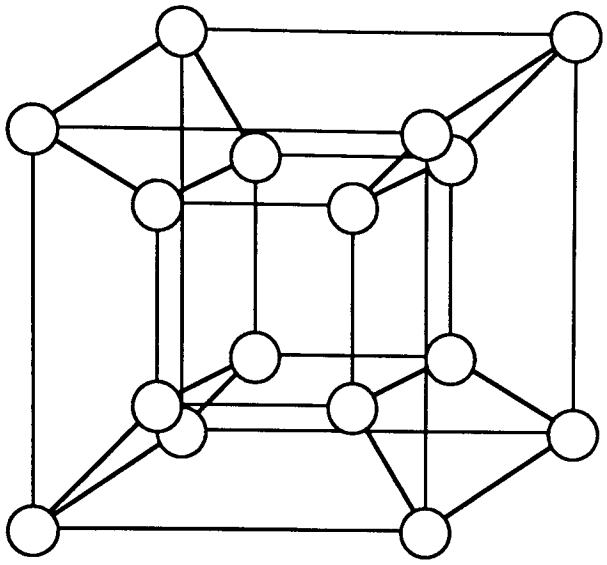


Figure 4.47 Distribution for male and female.

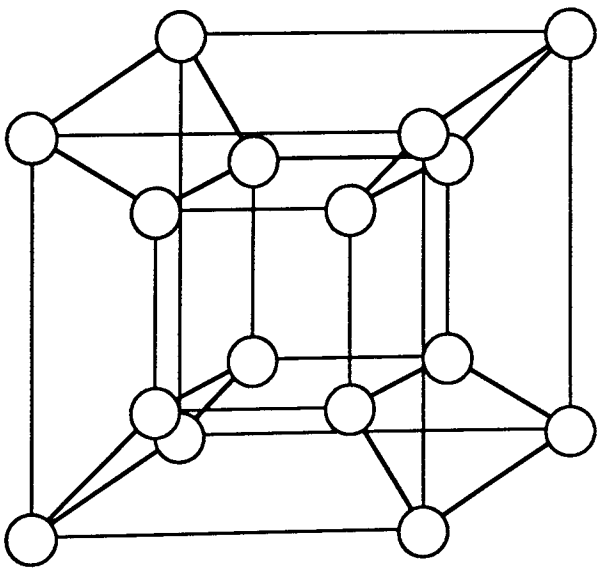


Figure 4.48 Distribution for blue and brown eyes.

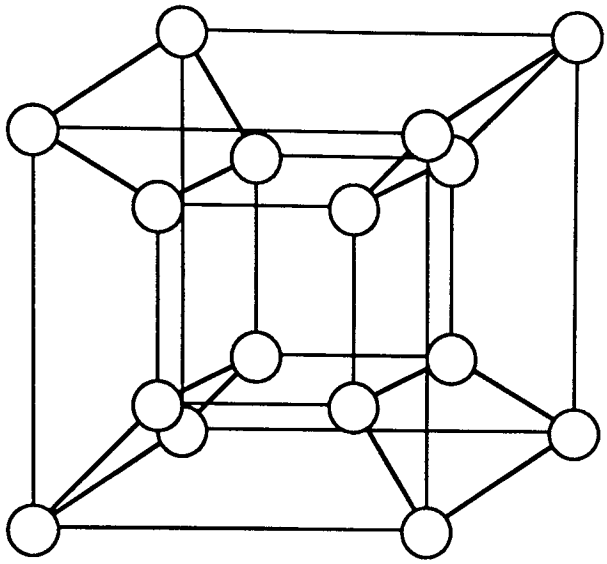


Figure 4.49 Training set and classification.

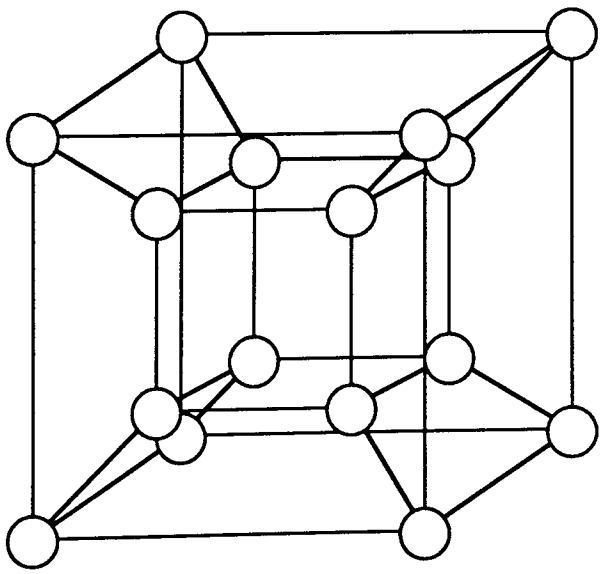


Figure 4.50 Result of the hypercube's generalisation.

4.4.2 Noise.

We have experimented with various types of pattern. For example, the "train problem" where east and west going trains (in stylised train drawings) are distinguished by a number of features such as: carrying a triangle, has black wheels, etc. The object is to describe the distinction in a minimum of rules. These intelligence test type of problems usually seem to present the machine with little difficulty. Patterns in the form of simple shapes such as Figure 4.51 are another useful source of experiments and the simpler ones were also used to test the machine's tolerance to noise. Specifying the pattern precisely seems to be the major difficulty. The machine will not act as expected if it is actually working on a different problem! (More automation would help). Simulating recognition of actual hypercolumn patterns (1 to 5) at the same (calculated) resolution works well. These are static striate cortex type patterns. Patterns such as 11 and 12 with nothing in common fail as expected. With some patterns the machine appears to display a logical rather than a learning mode response. The results occasionally seem surprising yet at the same time irrefutable. This needs further investigation as does its general pattern recognition capability. Our approach to noise tolerance was inspired by the work of Ehrlich, Crabtree and others in a series of papers (for example, Ehrlich et al., 1984). Their approach, used in petrographic image analysis, involves a *metrical* "erode and dilate", the effect of which being to eliminate noise by shrinking the field of view below the expected average threshold of noise but not sufficiently to lose the picture information. The field of view is then expanded back up to its original size whereupon the image is then "clean".

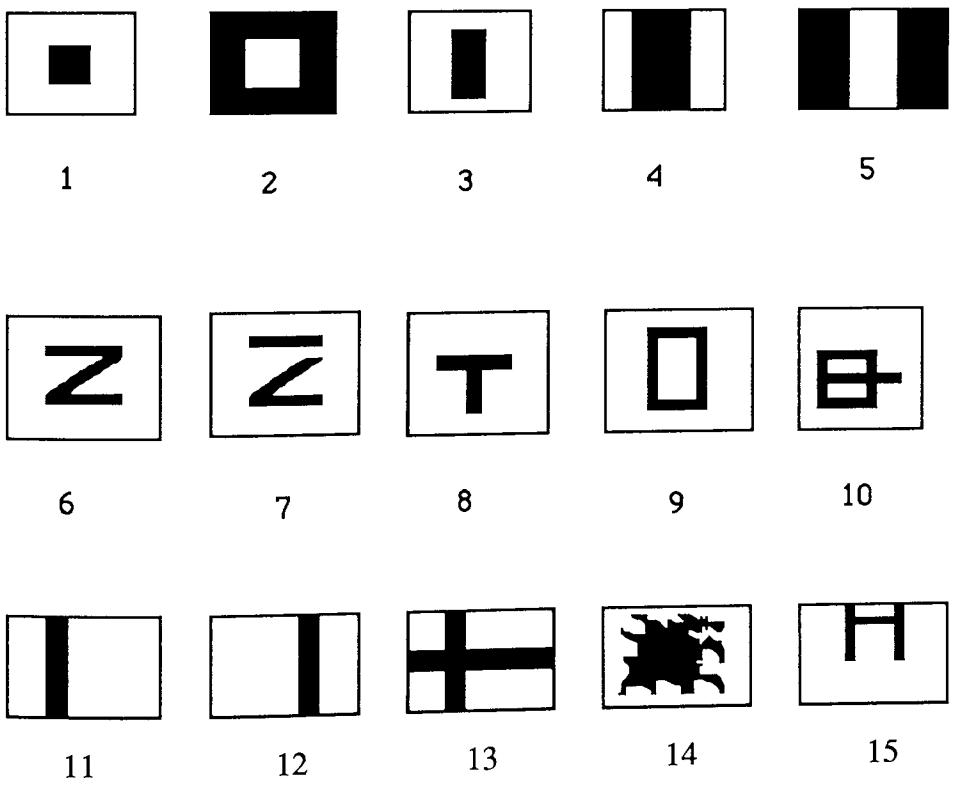


Figure 4.51 Typical simple pattern experiment shapes.

To save space we eliminated our noise theory from chapter 3. In the following we are therefore brief. We have implemented an analogous procedure in the subhypercube machine. Our quantisation threshold is boolean and quite crude being set at 1, 2, 4 or 8 (in decimals) for any given subhypercube. We have therefore been quite surprised to find how robust and reliable such a simple technique can be. The disadvantages of the method are firstly that a heavy price is paid typically in the intermediate noise levels in loss of speed. As the noise level increases or is set at a higher level the effect is like resolution, limiting the depth of search hence speeding up the algorithm again. This is seen in Figure 4.53. Secondly to work effectively the method demands than the minimum number of exemplars be increased from 3D (or 4D to be safer) to 3DN (or better 4DN) where N is the noise level. Table 4.12, Figure 4.52 and Figure 4.53 illustrate these points. For example, for the sixth dimension at noise level 2 learning took approximately 50 seconds. Yet at level 4, with much higher noise levels about 11 seconds was sufficient.

D	R	N1	N2	N3	N4
1	1	0.067	.	.	.
2	2	0.250	0.133	.	.
3	3	0.634	0.533	1.660	.
4	4	2.433	2.700	1.417	0.250
5	5	6.817	9.767	6.083	1.333
6	6	25.050	47.400	40.60	11.33

Table 4.12 Algorithmic timings: levels of noise and dimensionality.

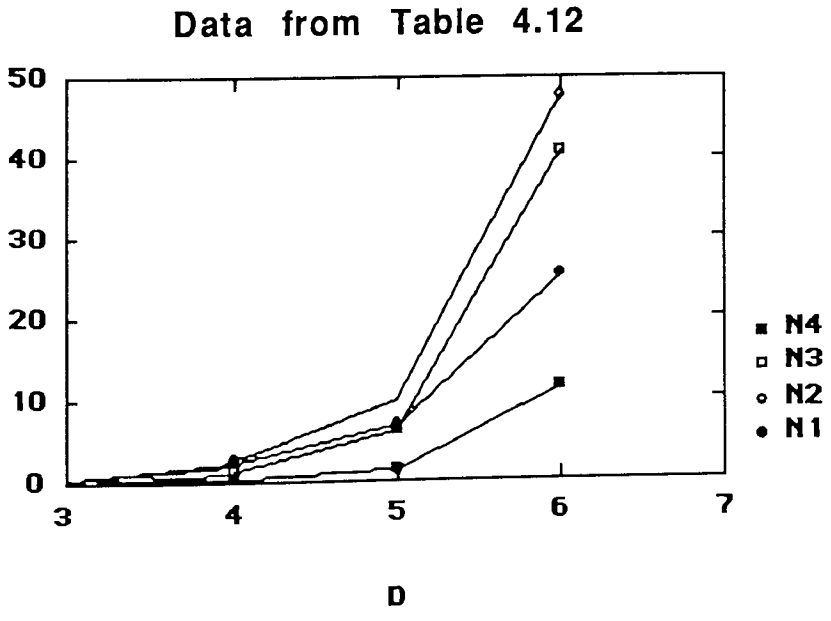


Figure 4.52 Noise immunity costing.

For further work: dimensional independence and considerable speed-up could be effected by simply making the method truly subhypercube based. A proper study of the hypercube machine's response to noise would, of course, demand a full Ph.D. project.

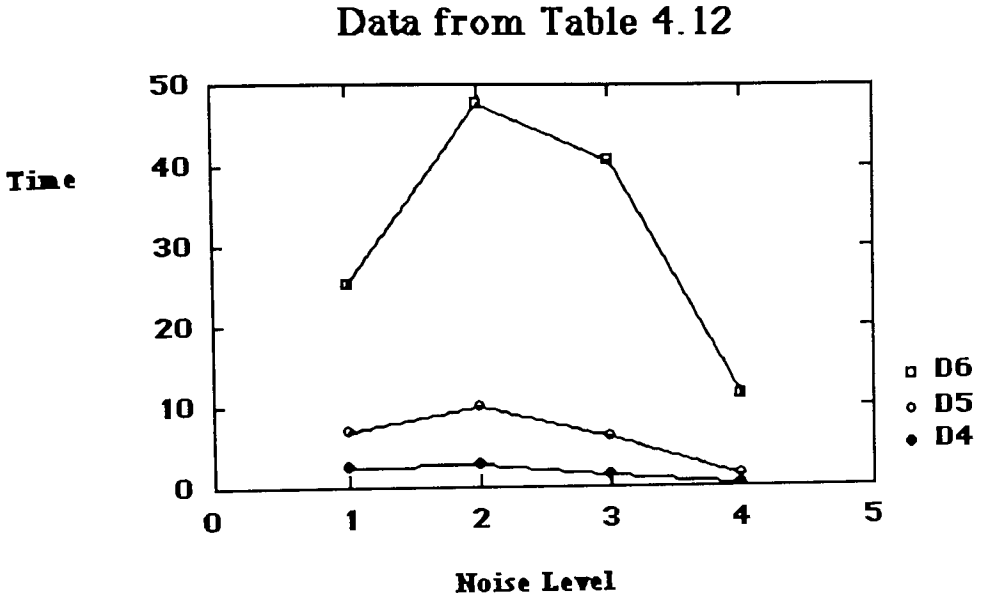


Figure 4.53 Noise level / dimensionality effect.

4.4.3 Limitations of the machine.

As we discussed in chapter 1 we originally decided to attempt to theoretically define a coalesced machine in order to combine features from several fields of machine learning. In retrospect we have made some progress we could considering the difficulty of the exercise. Our machine has some seminal, useful and novel features which, if we are to take our subject seriously, need to be fully understood. The same exercise also being required to explain weak points. At present much excellent prior A.I. work on representation etc. is being ignored in the new enthusiasm for neural nets. But in the longer term we would expect to see an increasing number of attempts at combining learning machines of various types. Our machine will be one of the forerunners of that future trend. So what mistakes did we make? What lessons can be learnt?

We find that there are a number of limitations on the machine, mostly stemming from decisions made much earlier on, typically in the earliest stages of the theoretical work. One big problem is an inability to represent more than a boolean choice for classification purposes. We originally took the view that since real neurons only have a single axon output it must be possible to solve most problems without the need for multiple classes. This is incorrect on a number of counts. As more is being learnt about the brain it is becoming apparent that a sizable percentage of neurons do not conform to this model of a classical neuron. Often there are multiple axons or even no discernible axon whatsoever.

Many problems that would allow comparison with other work are also not possible due to this omission. Many problems that could otherwise be attempted are ruled out. One major difficulty is that every new class that is added raises the complexity by a further factor of 2^n and we can ill afford that. Hence without a radical change in the theory no progress is possible in this regard. However, if complete convergence is not required there is one possibility. We can feed our multiple class outputs into the input side. This is not a procedure that we have had the time to theoretically investigate. It certainly works much of the time, but we have not established either theoretically or empirically how safe this procedure is. Firstly the reduction in the otherwise space of possibilities is really huge which implies that many functions may thereby become unlearnable or allow only the poorest of approximations. Secondly it is honestly an afterthought! Another difficult is that we still must provide a binary classification. This is only possible if we are able to specify invalid states in order to establish the requisite positive and negative classifications. But these invalid states are not necessarily knowable *in practice.*

Comparison with back propagation neural nets particularly as a result of comparative work on very simple parity problems suggests that the subhypercube groups or polytopes fairly directly correspond to a normalisation of the hidden and outer layer weights in these nets, at least, for parity functions. Yet the translation is non-trivial and corresponds to a mapping between continuous and discrete functions. A mathematical solution to this problem has been found as we discuss later and may well hold promise for extrapolating our research.

Domain independence has considerable advantages - the machine's usefulness is not restricted to just certain domains or problem types. Boolean microfeatures are almost infinitely flexible. Yet their very flexibility and uniformity at such a low grain size offers no insights with regard to general properties of the domain which may well be the key to much higher constraints which are more effectively able to limit the space. (An analogy is possible here with small grained rules and larger grained frames in A.I.). The resulting formulae are much easier to handle than decision tree outputs. But if the machine produces too many formulae either because the problem was poorly constrained or because the problem was hard then we may once again drown in a sea of information and the end result is no better.

Encoders are of considerable interest because they are able to form a basis for recirculation or unsupervised learning. The hypercube cannot perform this function since multiple classes are not available to mirror the image vectors. Feedback is also missing from the design of the machine. A further unfortunate omission. Competitive learning and spontaneous recovery are also outside the scope of our machine. The absence of the ability to handle real-valued inputs and outputs is also in practice much more restricting than expected. If we attempt to work with booleans and then compete effectively in practice with real-

valued machines then particularly at higher dimensions we are being over optimistic. It is possible to handle the figures, for example, by working within set ranges. But this is not a satisfactory solution. Lastly we require a more conventional structure of hidden layers etc. However, in finding ways to extend the theory to include some of these characteristics we must also retain our present hard won gains and **crucially important insights into the nature of learning**. Or else we start yet again "from scratch".

4.5 SPECIFIC ADVANTAGES OF THE MACHINE.

Our global subhypercubing learning machine has certain specific advantages when compared with other learning machines and in particular connectionist machines. The first of these advantages relates to its consequent reliability, dependability and reproducibility. In short the machine has a **predictable behaviour**. Same problem, same learning time, it never "gets lost". This covers another aspect of what we mean by the term "metric". (See "learning metric" in Appendix 2 and sections 3.4.2, 3.4.4, 3.4.5, 3.5.7, 3.5.7.1 and 4.3.2).

4.5.1 Orthogonality and Occam's Razor.

Let us illustrate this "**metrical**" progression of the algorithm by a very simple worked example. See Figure 4.54 Hopefully this diagram will impart to the reader something of the beauty of the mathematics inherent in the hypercube representation and its attendant subhypercubing algorithm. Remember that our learning machine is independent of domain coupling. It is, therefore, our belief that: the ***essence of machine learning at its most basic level*** is seen in the rhythmical wavelike progression of the algorithm; **metrically searching** at each level, learning by expanding and contracting subhypercubes accordingly. Let us see how it works.

In the following dissection all algorithmic details are ignored in order to highlight the larger picture. We will work through the case of a simple cube where opposing corners of each face are class '1', say, as filled circles in diagram (a) in Figure 4.54. The first task is to build a representation of the complete **universe** of possibilities. This is achieved using the dimensionality of the problem and is shown in the diagram as a variables only list thereby allowing for all combinatorics within the scope of the problem.

The "**universal hypercube**" is a cube itself in this simple case. This universal hypercube is also called the "**zeroth level subhypercube**". (This first list is actually a list containing a list. For shorthand we shall refer to such structures, in general being a 'list of lists of lists ...', as a "multilist"). This subhypercube, itself a hypercube, is then "**expanded**" using the supervars and templates, "t" database into all its possible

subhypercubes (the six faces of the cube). This expansion follows the particular **pattern order** of the 'gent' expansion as illustrated in Figure 4.54 (b) and numbered 1 to 6. Each of these hyperfaces is checked to see (a) if any are complete (all nodes are of the same class type) whereupon they are used to build the formula database, otherwise (b) the hyperfaces are complex (involving both boolean classes). The latter will require further expansion and constitutes the components of the second multilist in Figure 4.54.

In the case of the second multilist all possible subhypercubes exist in the multilist. (A strong indication of parity and an enticing pointer for further work). The multilist is checked for duplicates - in this case there are none. The last stage in the cycle is a "**reduction**" process whereby we wish to eliminate all sublists which do not contain any new information. This gives us the third multilist in Figure 4.54

Thus each cycle consists of three major phases: expansion, duplicates check, reduction. We have of course eliminated all detail in this description and further, for reasons of efficiency the three phases are not in practice as distinguishably sequential as described above but are instead interlaced. Nevertheless the above description catches the essentials of the algorithm.

We then repeat the process so that the fourth multilist is the result of the next expansion stage. Even for the cube we can see that the process has become quite complicated by the time we have reached this **second level subhypercube**. (For the second level remember that a square is composed of four lines). The fifth multilist is the result of eliminating duplicates and the sixth multilist the result of the reduction process. There are then four sublists each with one unknown. These lists are unique and can already be seen to be targeting upon the four parity nodes in Figure 4.54 (a).

In order to obtain the final nodes the algorithm expands the sixth multilist into the **third level subhypercubes** which is the seventh multilist, eliminates duplicates to produce the eighth multilist and reduces the information to produce the nineth multilist. This last multilist is simplified to reduce the redundant multilist structures to give the tenth multilist. The components of the tenth multilist are seen to correspond to the original nodes of the problem in Figure 54 (a).

This is highly unusual and only occurs near parity (where the critical range is small). In the general case (in practise, very close to 100 % of cases) the final multilist contains the ***generalisation*** of the exemplar information. (Parity, however, is so nonlinear as to demand rote learning. It is the essence of parity that it contains no generalisation only specialisation; which is why it is regarded as the ultimate benchmark for machines specialising in generalisation). The tenth multilist then contains the **formula** or (in the general case) **generalisation of the data.**

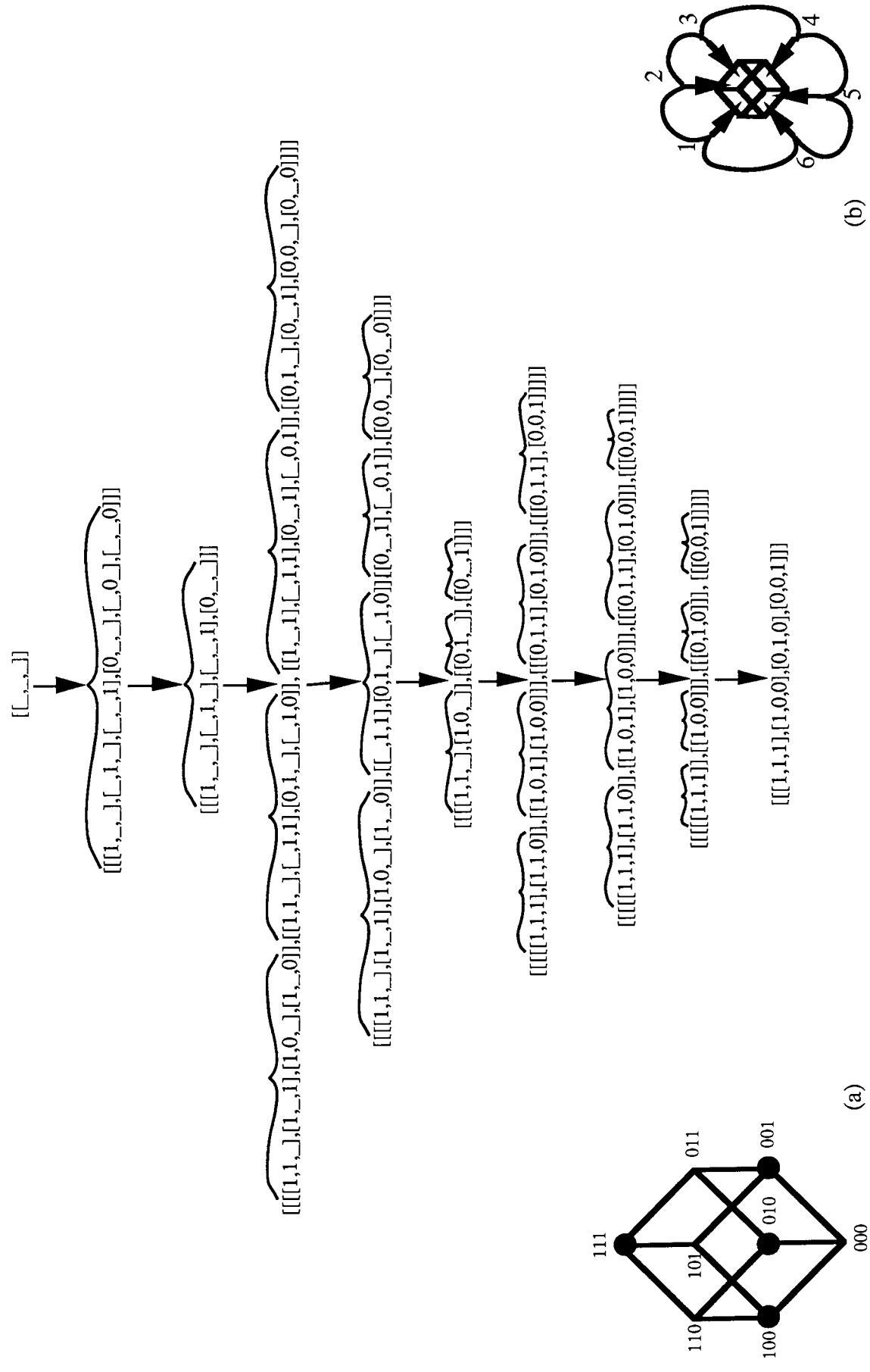


Figure 4.54 An example of the algorithmic process in action.

4.5.2 Depth Resolution.

We have mentioned the term resolution a number of times. In particular in section 4.2.4.4 we stated that resolution, R, sets a limit on the search depth and is used when the complete solution is not required. As R tends to the parity distance of the universe the formulae tend to 100% universal accuracy. The usefulness of depth resolution is, as one might expect, problem dependent. In favourable circumstances (large parity distance i.e. tending to linearity) and when the dimensionality is high, then first level resolution will provide approximately a 2*D division of the problem (compare section 4.2.2.2). For example, if D = 50 then we may attain 1/100 = 0.01 error or 99% accuracy - without the need to search further. Because resolution allows much faster processing, this facility is a significant novel contribution to exceptionally fast learning.

Let us consider how this works in practice. The algorithm allows the search to be stopped at the end of any given level and the formulae are then produced approximating the final solution. (Stopping in the middle of a level makes no sense). Clearly when the resolution depth equals the dimensionality we have the conventional case that we have seen so far. The advantage of depth resolution is that an approximate solution may be produced in an acceptable learning time even for large problems.

4.5.2.1 Resolution guidelines

This means that in practice it is useful to provide guidelines to the user on what is the maximum resolution depth that would lead to an acceptable solution within a very short time period (a few seconds for D < 100). The (provisional) rules that we have used in the program to define the recommended value for R_{max} given different dimensions, D, in order to obtain the result in reasonable time, even if the problem is hard (nonlinear), are:

1 $0 < D < 16 \Rightarrow R_{max} = D$

2 $15 < D < 30 \Rightarrow R_{max} = D - 2(D - 15)$

3 $D > 29 \Rightarrow R_{max} = 1$

These recommended values illustrated in Figure 4.55 can, of course, be exceeded if the user wishes.

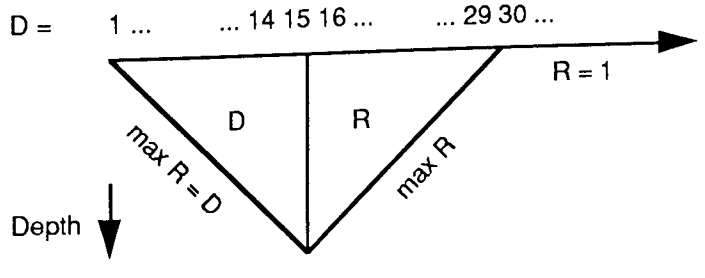


Figure 4.55 Recommended maximum resolution depths.

We will take a simple example to illustrate depth resolution, see Table 4.13. This tabulates the times for various resolution depths for various dimensions. Note that the problem needs to be the same in each case if we are to compare the results. For this purpose we use our standard benchmark (section 4.2.2).

D	R1NOLIST	R1	R2	R3	RD
1	0.067	0.067	na	na	0.067
2	0.067	0.067	0.250	na	0.067
3	0.067	0.067	0.317	0.634	0.067
4	0.083	0.083	0.450	1.183	2.433
5	0.100	0.100	0.517	1.950	6.817
6	0.116	0.134	0.850	4.483	25.050

Table 4.13 Depth resolution.

Note: 'na' in Table 4.13 means "not applicable". We cannot have a resolution greater than the dimensionality (since a tree structure cannot be searched beyond its base). This understood, we take the liberty of occasionally introducing "dummies". Dummy => na - guesstimated even though it does not exist, in order to help complete certain graphs where otherwise initial values would be missing leaving fewer points on some of the curves, thereby disabling comparisons. Thus the dummies are for illustration purposes only.

A statistical analysis of Table 4.13 is given in Table 4.14. Clearly these results provide evidence of a rapid change in the nature of the curve as the resolution is increased from R = 1 to R = D. This we can further illustrate by *normalising* and putting all the resulting resolution graphs together in a "splom graph" as Figure 4.56. The reader should pay particular attention to the way in which the last point on each graph relates to its predecessor as the resolution increases (looking down the graphs). Clearly, the the graph is changing from linearity to non-linearity. We should remind the reader that first level resolution was theoretically predicted to be linear and its practical embodiment shows evidence for this prediction.

Descriptive Statistics: Total observations: 6

N of cases	6	6	5	4	6
Minimum	0.067	0.067	0.250	0.634	0.066
Maximum	0.116	0.134	0.850	4.483	25.050
Range	0.049	0.067	0.600	3.849	24.984
Mean	0.083	0.086	0.477	2.063	5.875
Variance	0.000	0.001	0.055	2.895	94.654
Standard dev	0.021	0.027	0.234	1.702	9.729
Sum	0.500	0.518	2.384	8.250	35.250

Table 4.14 R1-D Statistical Analysis.

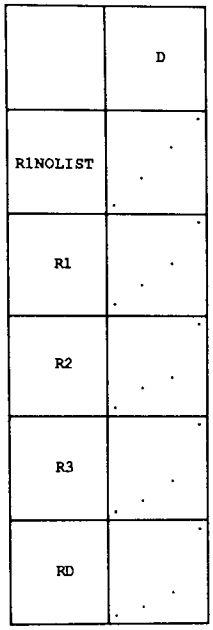


Figure 4.56 Comparing explosions by splom.

The best way to see just how effective depth resolution is in fighting the exponential explosion is via a series of graphs as Figures 4.57 to 4.59 inclusive. In Figure 4.57 the change in slope at D = 3 should be disregarded since it results from extraneous factors (the noise being greatest for the first two dimensions, etc.).

R1NOLIST in Table 4.13 refers to an attempt to show that as the dimensionality increases there is no new detectable explosion from the increasing size of the output listing itself. (The reader will remember this inversion is proper, section 3.1). This is shown to be incorrect since the graphs for 'R1' and 'R1 with no output listing' diverge. The multiple effects of minor details such as this shows just how difficult it is to squeeze out all the additional sources of explosion.

In Figure 4.58 we plot the results for all runs of less than 3 seconds. The graph is sufficiently large scale to allow us to see the way the explosion depends upon the depth of search. The graph is seen to divide into 5 definite curves. (Again the first few points have much higher associated error).

As longer times are considered in Figures 4.59 we can see very clearly the larger picture (at the expense of the detail of Figure 4.58) of just how effective depth resolution is in overcoming the complexity problem. The search explosion can be contained even further by parametrically allowing certain other constraints.

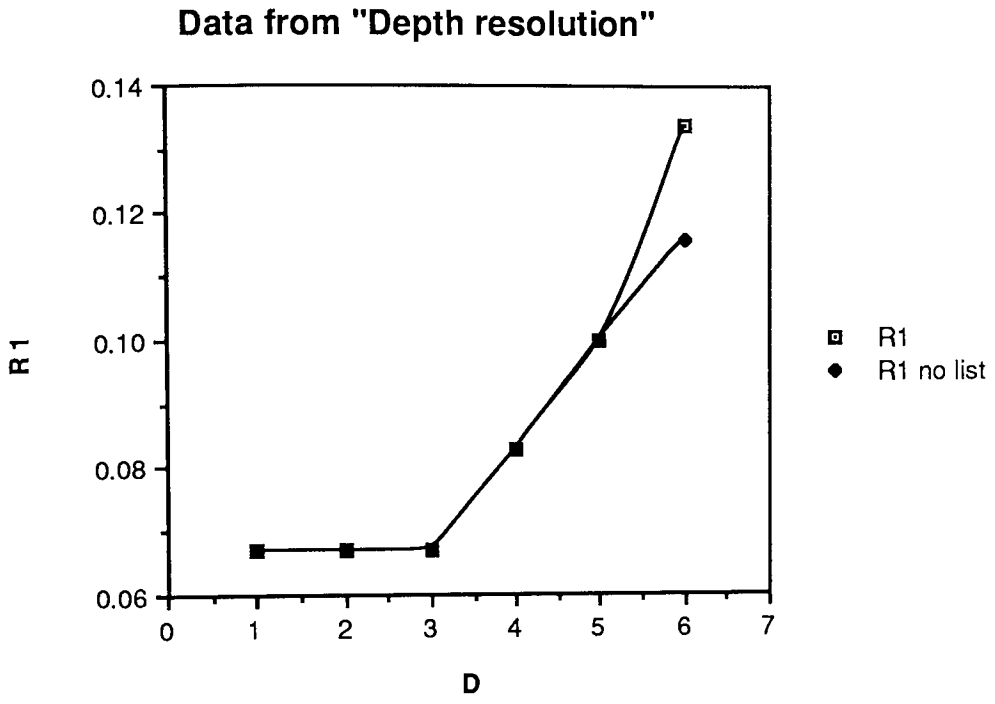


Figure 4.57 The output loading effect.

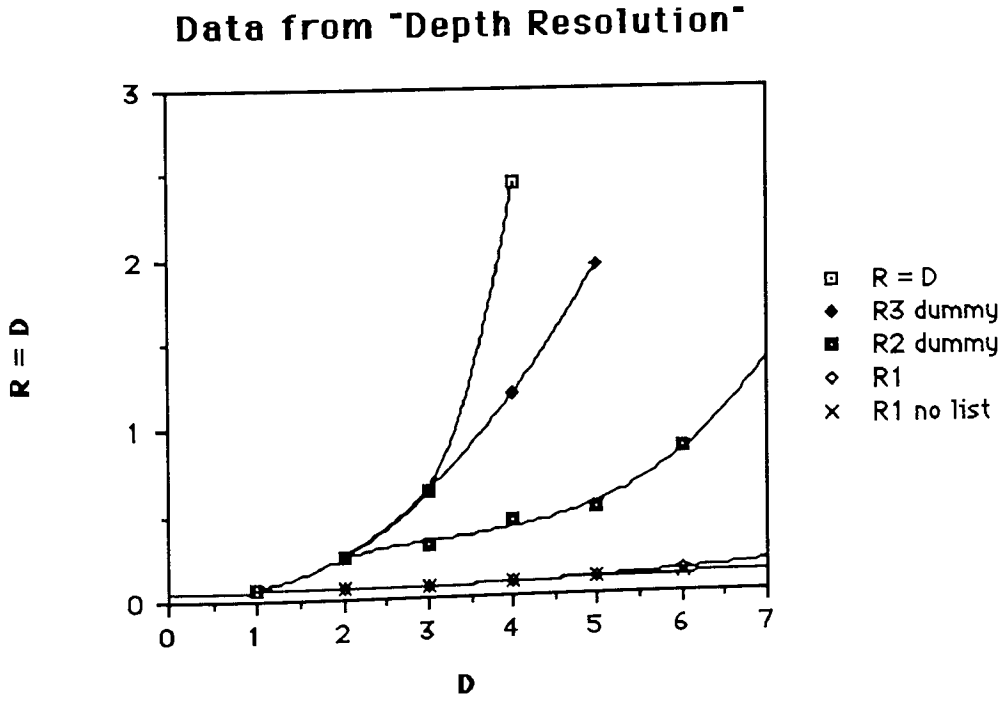


Figure 4.58 Resolution R = 1 to D low.

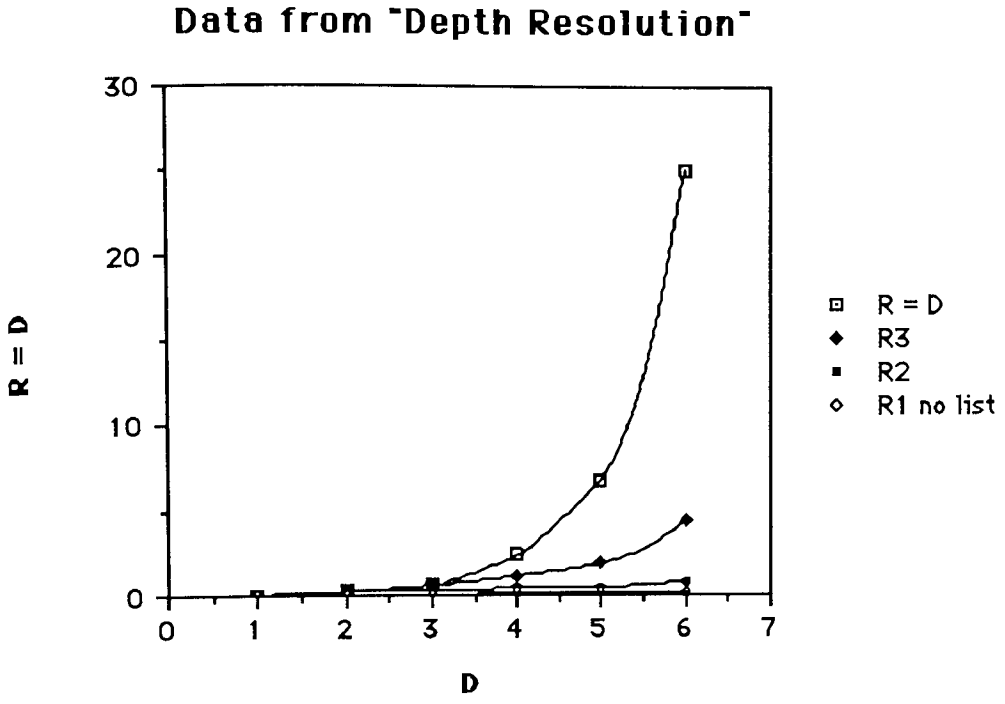


Figure 4.59 Resolution R = 1 to D high.

We conclude this section with our earliest evidence of the learning curve for a typical problem where R = D/2. In general we find that essentially the curve shifts to the left for increased linearity, larger values of D, larger values of R and shifts to the right for increased nonlinearity, smaller D, smaller R - if other factors are kept constant.

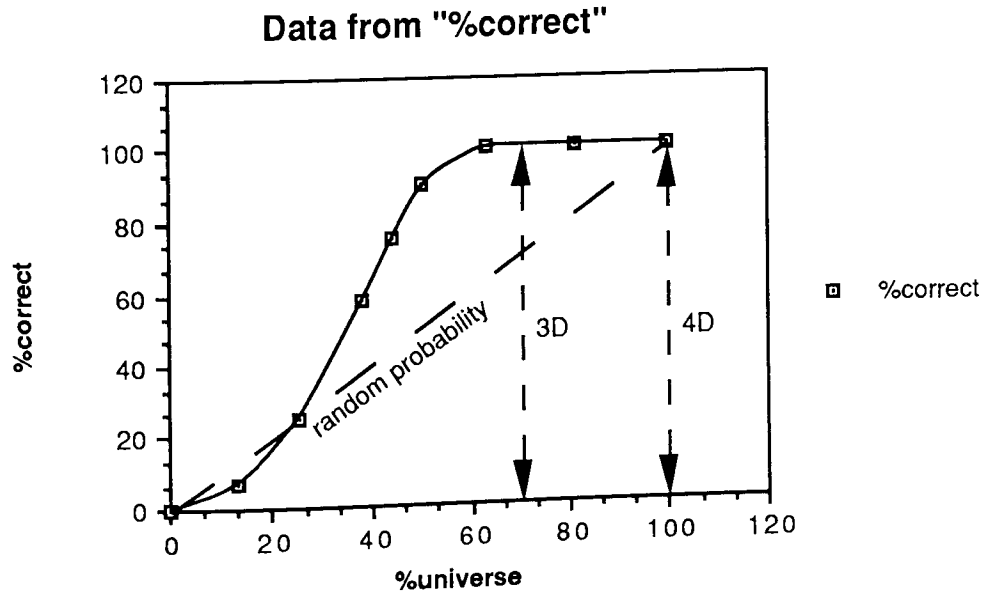


Figure 4.60 The learning curve.

4.5.3 Top Bot.

"Top bot" refers to a test of the top and bottom node of each subhypercube. Since any given node is connected to half of the complete subhypercube there is an opportunity to speed up the search considerably. The advantage occurs every other dimension all other things being equal this leads to a slight leapfrogging effect for parity which is just about visible in Figure 4.61, based on Table 15 when top bot is true. In contrast Figure 4.62 illustrates the case for top bot false. Notice the slow advantage accruing via the leapfrogging effect by comparing Tables 15 and 16. E.g. 2x = 0.05 in both tables, speed-up affects only odd dimensions).

D	T1	T2	T3	T4	T5	ln(T1t)	ln(T2t)	ln(T3t)	ln(T4t)	ln(T5t)
1e	0	0	0	0	0	-4	-4	-4	-4	-4
1o	0	0	0	0	0	-4	-4	-4	-4	-4
2e	0.05	0.05	0.05	0.033	0.033	-3	-3	-3	-3	-3
2o	0.05	0.05	0.05	0.05	0.05	-3	-3	-3	-3	-3
3e	0.133	0.134	0.133	0.133	0.15	-2	-2	-2	-2	-2
3o	0.117	0.15	0.134	0.133	0.15	-2	-2	-2	-2	-2
4e	0.534	0.55	0.533	0.55	0.533	-1	-1	-1	-1	-1
4o	0.467	0.45	0.467	0.45	0.45	-1	-1	-1	-1	-1
5e	2.2	2.2	2.2	2.2	2.2	0.788	0.788	0.788	0.788	0.788
5o	2.234	2.2	2.217	2.2	2.183	0.803	0.788	0.796	0.788	0.780
6e	13.25	13.65	13.7	13.717	13.734	2.583	2.613	2.617	2.618	2.619
6o	11.617	12	11.95	11.983	11.983	2.452	2.484	2.480	2.483	2.483
7e	89.816	91.8	91.916	91.7	91.333	4.497	4.519	4.520	4.518	4.514
7o	89.167	90.666	91.15	91.25	91.15	4.490	4.507	4.512	4.513	4.512
8e	761.983	864.4	847.333	851.05	850.55	6.635	6.762	6.742	6.746	6.745
8o	724.983	817.483	814.217	812.7	799.066	6.586	6.706	6.702	6.700	6.683
9e	7336.917	7585.35	7533.117	7533.967	7533.967	8.900	8.933	8.927	8.927	8.927
9o	7464.5	7574.167	7533.25	7506.45	7584.983	8.917	8.932	8.927	8.923	8.933

Table 4.15 Top bot true results.

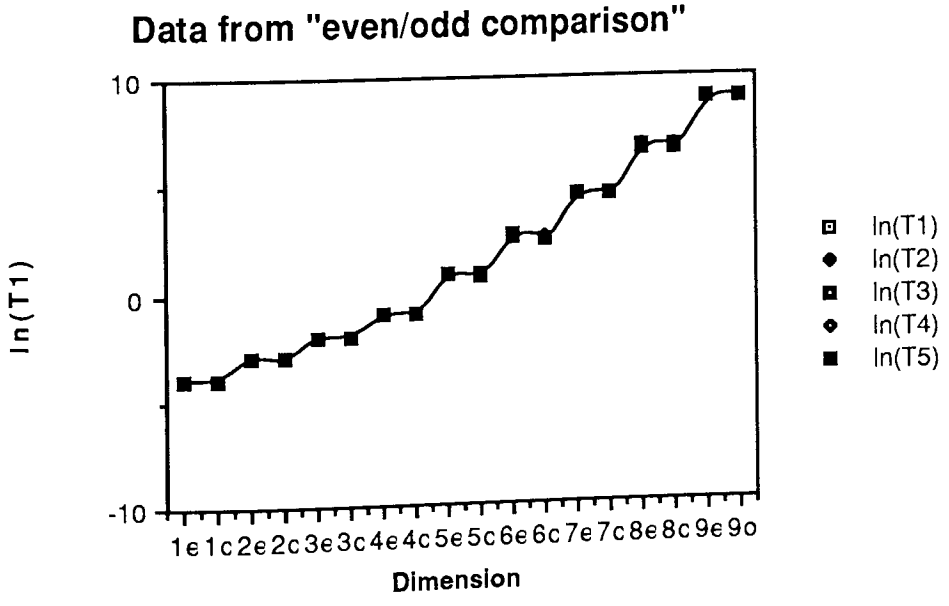


Figure 4.61 Top bot true.

D	T1	T2	T3	T4	T5	ln(T1f)	ln(T2f)	ln(T3f)	ln(T4f)	ln(T5f)
1e	0	0	0	0	0.017	-4	-4	-4	-4	-4
1o	0	0	0	0	0	-4	-4	-4	-4	-4
2e	0.05	0.05	0.05	0.05	0.067	-3	-3	-3	-3	-3
2o	0.05	0.05	0.05	0.05	0.067	-3	-3	-3	-3	-3
3e	0.183	0.2	0.183	0.167	0.2	-2	-2	-2	-2	-2
3o	0.183	0.167	0.183	0.183	0.167	-2	-2	-2	-2	-2
4e	0.817	0.783	0.817	0.783	0.733	0	0	0	0	0
4o	0.833	0.784	0.85	0.766	0.8	0	0	0	0	0
5e	3.95	3.966	4.083	4.034	3.983	1.373	1.377	1.406	1.394	1.382
5o	3.95	3.933	4.017	3.966	3.9	1.373	1.369	1.390	1.377	1.360
6e	24.9	24.617	24.567	24.784	24.8	3.214	3.203	3.201	3.210	3.210
6o	24.416	24.45	24.65	24.584	24.733	3.195	3.196	3.204	3.202	3.208
7e	192.283	193.617	194.434	192.85	193.283	5.258	5.265	5.270	5.261	5.264
7o	192.15	193.716	192.95	192.884	192.8	5.258	5.266	5.262	5.262	5.261
8e	1864.283	1814.516	1867.016	1820.1	1824.05	7.530	7.503	7.532	7.506	7.508
8o	1863.317	1816.867	1865.7	1816.484	1823.4	7.530	7.504	7.531	7.504	7.508
9e	17018.467	16908.583	16962.75	16951.85	16943.617	9.742	9.735	9.738	9.738	9.737

Table 4.16 Top bot false results.

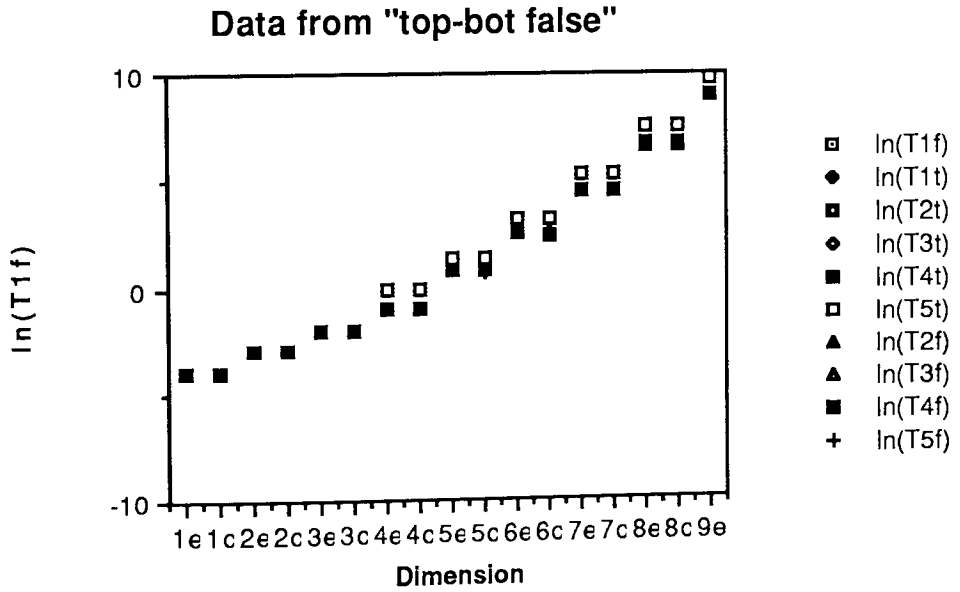


Figure 4.62 Top bot false.

4.5.4 Binary chop.

Binary chop takes advantage of the fact that all nodes in the subhypercube appear in any two opposing hyperplanes, e.g. [1, ...] and [0, ...]. Binary chop, however can only be used to obtain approximate solutions. Considering a similar set of experiments on the Binary chop method as Figure 4.63 we can see a very significant speed-up. Also the explosion using Binary chop is very much reduced, as expected.

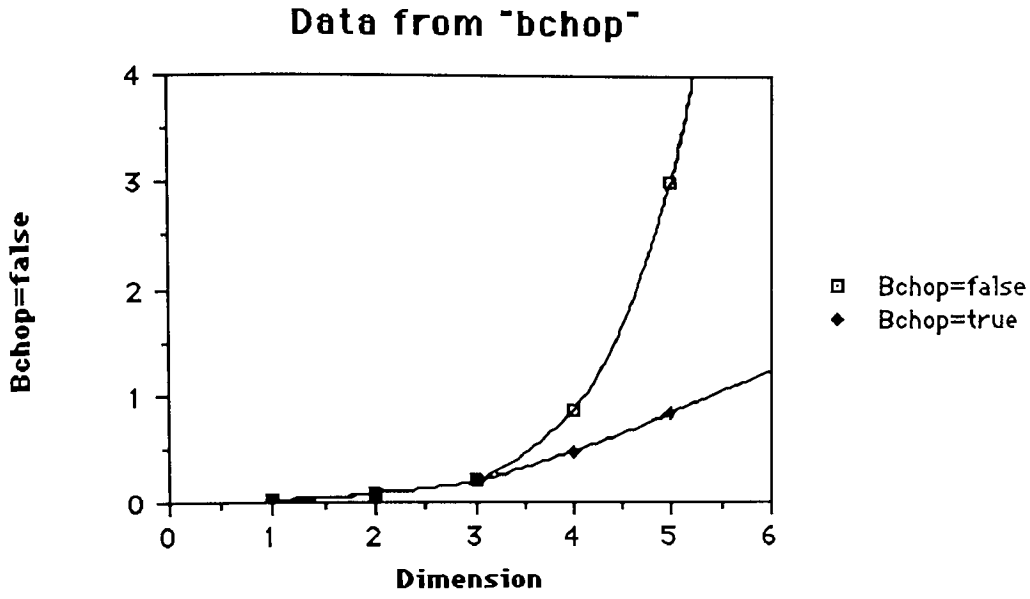


Figure 4.63 The effect of binary chopping.

Avoidance of duplicate subhypercubes was eventually solved most elegantly by a procedure called "makesupervars". Makesupervars stands for "make super variables" and is a machine to generate appropriate super variable or "supervars" templates for a given level. Thus it is most infrequently used, only once per level. It avoids unnecessary duplication of effort. Supervars are a fast database template which enables very rapid calculation of how to eliminate one boolean from the list and recombine the remainder of the list into a new list. Expensive "concs" are therefore avoided within levels and restricted to between levels. All this being achieved in just one goal - hence use of the term "elegance". The supervars format is :

supervars(*Variables_position, Variables_old, Variables_new*).

The result was a significant effect on the speed-up, as expected. However, despite the excellence and satisfaction of this procedure, it was something of a disappointment in comparison with the change from recursion to iteration.

Lastly, a useful little experiment tested for 2nd order combinatorial effects using binary chop and learn with parameters as below, see also Table 4.17

Parameters:

D=R, N = 1, Binary Chop = **true**, Top_bot = false, Debug = false, listingformula = false, program = **learn**

D	Times	multiplication speed	multiplication acceleration
1	0.000	–	–
2	0.050	–	–
3	0.100	2.000	–
4	0.183	1.830	0.920
5	0.366	2.000	1.090
6	0.767	2.100	1.050
7	1.950	2.540	1.210
8	5.500	2.282	1.110
9	19.350	3.510	1.240
10	73.583	3.800	1.080
11	300.217	4.080	1.070
12	1204.267	4.010	0.980
13	4966.234	4.120	1.030

Table 4.17 Binary chop learn second order effects.

We define the multiplication speed, $MS = D_{n+1} / D_n$
and multiplication acceleration, $MA = MS_{n+1} / MS_n$.

The multiplication acceleration is a scattergraph with horizontal line at approx at **1.1** in Figure 4.64. Thus at last, in the case of the learn algorithm there are no 2nd order combinatorial effects noticeable here.

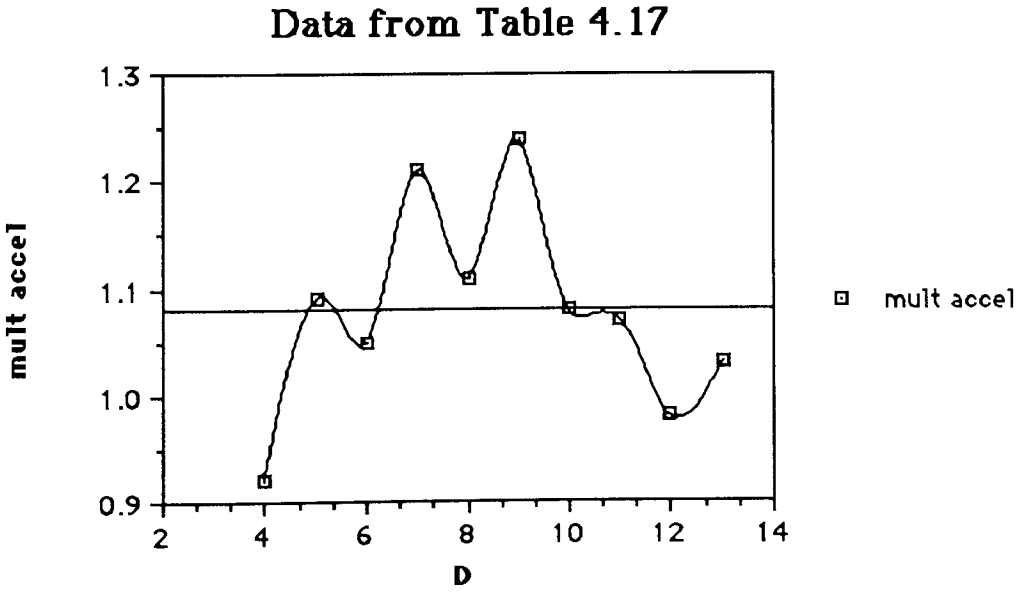


Figure 4.64 Second order multiplication effects.

4.5.5 Comparing machines.

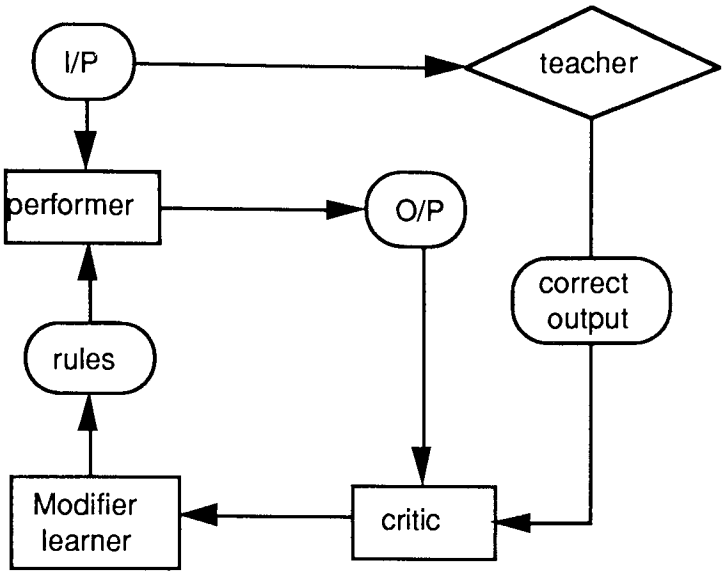
Firstly it is useful to compare the hypercube with other learning machines in as many ways as possible, since the comparisons can be instructive. We therefore provide in a number of ways further comparisons between our subhypercube machine and other machines. In section 2.3.4 we introduced Hypersphere Classifiers. In Table 4.18 we compare the subhypercube and the Hypersphere Classifiers along various crucial dimensions.

In the second comparison, Figure 4.65 compares the conventional machine learning cycle (a) with the subhypercube machine's learning cycle (b). It is seen that the differences are in detail rather than structural.

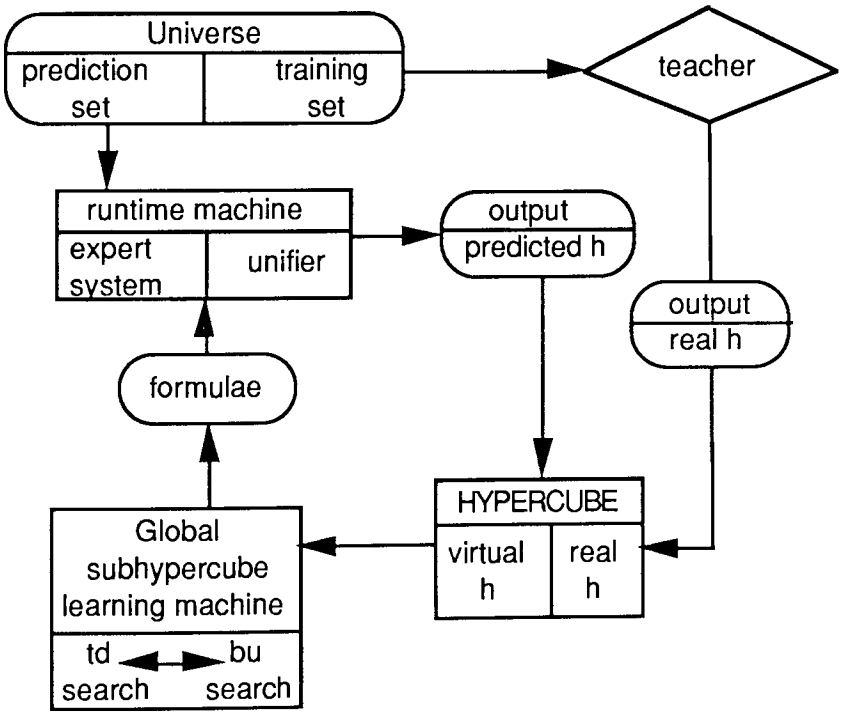
Thirdly, the graphs in Figure 4.66 and Figure 4.67 compare the performance of the subhypercube machine to a specifically optimised Error Back Propagation learning machine constructed by Andy Lowton at Aston university and used for a considerable number of parity tests (Lowton, 1992). The same machine was used for this BP machine and the subhypercube machine, namely, the sun SPARCstation 1. The same operating system was used (sun's UNIX system). The same external environment, networking loading, same time of day etc.

	Hypersphere classifiers		Subhypercube algorithm	
Decision surface	nonlinear	✓	nonlinear	✓
Decision surface Placement	heuristic	✗	metrical (deterministic)	✓
Produces degenerates?	yes	✗	no	✓
Exemplar limited?	yes	✗	no	✓
+ve/-ve bias	asymmetrical	✗	symmetrical	✓
Noise tolorant?	weak (statistical)	✗	good (metrical)	✓
Perceptron comparison	>	✓	>	✓
Multiple classes	weakly extensible	?	no	✗
Input/output	real-valued	✓	boolean	✗
Score =	3		7	

Table 4.18 Comparing the Subhypercube and Hypersphere Classifiers.



(a) The classical machine learning cycle



(a) The subhypercube machine learning cycle

Figure 4.65 Comparing the classical and subhypercube learning cycles.

The languages used were different. The hypercube is written in an operationally very slow but very high level language, QUINTUS PROLOG. Andy's BP is written in modula 2. The graphical comparison gives the BP machine an order of magnitude or so advantage in that it was not run to 100% accuracy - that would take too long. Even so, the subhypercube machine is seen to be not only significantly faster but even more so with increasing dimensionality.

The reader may wonder what all the fuss is about parity, since a trivial counting program can solve the parity problem for the thousand'th dimension in mere milli-seconds!! The point is that is **not** the point! We are *not trying* to solve the parity problem. That is easy. We are simply using parity as a means of benchmarking *learning* engines. For that reason the gap between the above results is very significant. (The reader will forgive us if we labour the point).

Andy Lowton's back propagation results used for these comparisons are given Appendix 9. Other details of this machine and the details of other experimental research on parity undertaken by Andy are given in Lowton, 1992. Comparisons have also been made with the parity experiments of Sarbjit Singh Sarkaria with very similar results. Sarbjit's machine was again an optimised BP machine reducing the sigmoid towards a hard limiter to gain speed. This work is referenced in Sarkaria, 1990. The indications are that these machines are close to the best that has been achieved by any BP machine. The interested reader is referred to the work of Scott Fahlman on increasing BP speed on the parity benchmark, Fahlman, 1989. Also see Tesauro and Janssens, 1988.

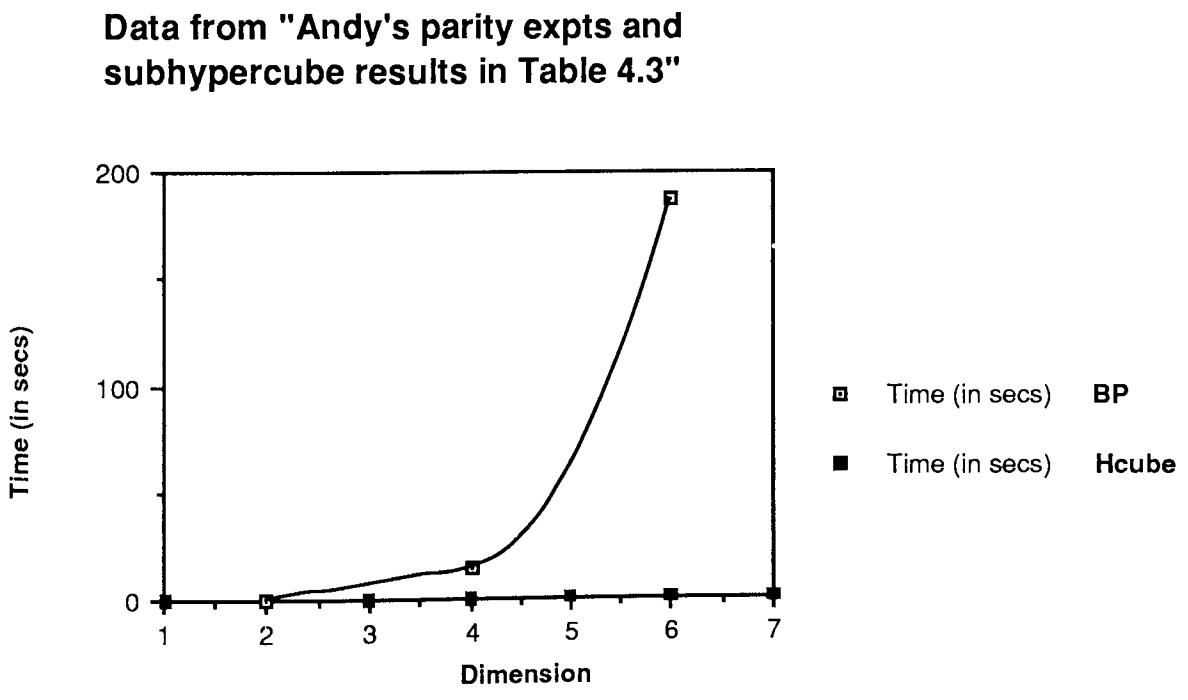


Figure 4.66 Exponential comparison between BP/subhypercube.

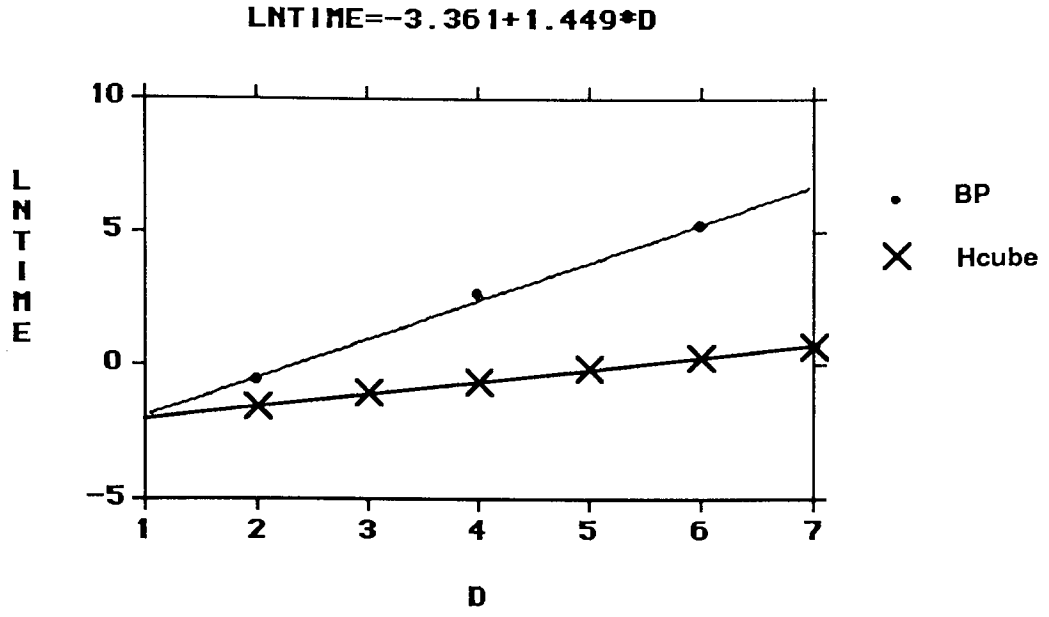


Figure 4.67 Logarithmic comparison between BP/subhypercube.

4.5.6 Some general heuristics.

As a result of various types of experiments undertaken and from the exercise of developing the learning machine certain lessons in fighting the complexity explosion have been learnt. Some of these general heuristics, often at the algorithmic metalevel, may be of interest to the reader, or at the very least amusing.

1 If a goal/task is repeated beyond a certain threshold number of times then consider storing and referencing it as required. (E.g. supervars).

2 Be aware of information implicitly stored in the structure of the algorithm itself.

3 Be aware of information implicitly stored in the metaprocesses of the algorithm.

4 Study where, when and why information is discarded (and possibly too soon).

5 Cleverer insights lead to faster algorithms - intelligence definitely pays off.

6 The cleverer the speed-up attempted in some segment of the algorithm the more code it will probably require and beyond a certain threshold the latter outweighs the former.

7 Unsubtle tools readily increase speed and decrease flexibility.

8 Subtle modifications less readily affect either speed or flexibility.

9 Considerable elegance and purchase results from deep rethinks which only handle the general case. And enhancingly, the special case then naturally finds its place.

10 A facility to visualise in the higher dimensions (at least D = 8) is *essential*, whereupon the insight becomes transparently obvious in the second dimension.

11 Bright ideas almost always work - and almost always have negligible effect.

12 Flashes of insight and inspiration come very easily. But only 1% of good ideas *are* good ideas. Eddington was right about the "perspiration" required.

4.6 AN ADVANCED APPLICATION OF THE MACHINE.

As an advanced application of the machine we will consider a real world problem. The problem is concerned with the use of neural net and expert system techniques for the design of drugs. Many millions of pounds are spent each year on the search for new drugs. Clearly, the pharmaceutical industry is exceptionally interested in the search for such techniques. This research is a joint venture between Professor Bodor together with his team at the Centre for Drug Design and Delivery, University of Florida, U.S.A. and Dr. Alan Harget at Aston University, and is based upon an original idea by Dr. Harget.

4.6.1 Prediction of the partition coefficient of compounds.

Essentially, the concern is with the prediction of the solubility and partition coefficient of compounds. Drugs need to get to their active target site easily, which implies that their mobility and solubility is important. It is found that the partition coefficient acts sufficiently as an approximation to these factors. Hence given an unknown drug the aim is to predict the log. partition coefficient and so determine the likely mobility of the drug in the body. The experimental data being used for this work was found to be very noisy and difficult to obtain because of its commercial sensitivity. The data used in this study has 17 parameter inputs and 1 output value all of which are real valued. The training set consists of 306 compounds and the predictive set contains 21 'unknown' compounds.

It has been found so far that neural net techniques using the back-propagation algorithm are more accurate than the technique of regression analysis on the training set; and reversely so for the predictive set, where at present it is weak. B.P. can reach an *average* accuracy of 90% in 2 to 3 hours of learning time. The eventual aim of this research is to be able to input the drug structure to possibly a number of neural nets which are associated with a knowledge base of pharmacological knowledge so as to predict its likely activity. An expert system would then produce the compound profile e.g. stable and toxic, etc. Harget and Bodor's work described above outlines a large very ambitious project which clearly will take many years. We for our part have attempted to apply the hypercube to this problem and our results have been encouraging for several reasons.

1. **The hypercube is able to cope with the problem size** as evidenced by preliminary results. An example printout in trace mode is given in Appendix 8 and is a **sixty-eighth dimensional problem.** This is significant!

2. Processing times on this preliminary work are **fast.** (Learning times of 3.540 to 12.560 seconds for a combination of resolution and Binary Chop settings within the algorithm).

3. **The problem is real world!** It contains data of large dimensionality and considerable experimental noise.

4. **This problem was totally unseen.** We have developed the machine completely independently of any target drugs problem. Indeed the drugs problem was mere chance - we could have attempted some completely different problem. We believe that it works *because* it is domain decoupled, and *it worked first time* without the need for *any* code modifications to accommodate the domain.

This is a testimony to the generality of the approach and the robustness and flexibility of the program. There follows the contents of the output file "out68_1" from our first attempt to apply the hypercube to the drugs problem:

% ********** Program: learn **********
% FILES used were:
% Output filename = out68_1
% t input filename = gent68
% Database input filename = alans_drugs
% PARAMETER settings were:
% Dimension = 68
% Resolution = 1
% Noise = 1
% Binary Chop = true
% Top bot = false
% Debug = false
% Listing formula = false
% Estimated maximum processing time 0.0566666 minutes
% SEARCHING levels:
% Generating level 0
% Generating level 1
% Learning took 8.150 sec.

A word of caution. It is true that one could very easily spend several years investigating the efficacy of the hypercube with respect to the drugs problem and against other approaches. Further work clearly needs to be done on real world problems so that more detailed knowledge of the scope and performance of the hypercube approach can be ascertained. It is encouraging however that the effectiveness of the hypercube in dealing with an advanced real world problem has already been demonstrated - even though the machine has been developed "from scratch"! Our results, however, for the present, must remain *very preliminary*.

4.7 REVIEW.

We will now review this chapter under four major headings. First, we provide a summary of this chapter. Second, we provide a discussion of the work with particular reference to this chapter. Thirdly we outline suggestions for further work. Fourthly, we offer our conclusions on the project as a whole.

4.7.1 Summary of chapter 4.

Since the major sections of this chapter were each purposefully concerned with very distinct aspects of the experimental work as prescribed in the introduction, it is most efficient to summarise these sections separately as follows.

4.7.1.1 Summarising section 4.1

We began this chapter by stating that we have developed a novel machine learning theory (as described in Chapter 3) and this degree of novelty in the project thereby, somewhat counter-intuitively, necessitates more not less experimental work in order to justify the machine. To summarise section 4.1, we identified five requirements as the minimal necessary work to obtain the requisite credibility. These were:

1) Development of the subhypercube machine.
2) Testing the predictions of the theory.
3) Scope and limitations of the machine.
4) Specific advantages of the machine.
5) An advanced application of the machine.

In **"preliminary considerations"** (section 4.11 and ensuing sections) we reported our initial attempts to develop a hypercube-based machine: these were the Karnaugh mapping and the "specific to general" approaches. We also discussed the considerable experimental difficulties of taking reliable readings, particularly in the earlier stages of the project. Two further points were firstly the important comment that: "The subhypercubing algorithm is theoretically **deterministic** for finite hypercubes with defined start and end points in its convergence and cannot, therefore, "get lost" or enter an infinite loop" (section 4.1.1.2). Secondly we also suggested that: "Simple comparisons between QUINTUS and C suggest a possible speed-up by recoding of up to 10^3 times. This could turn the work into a valuable fast product and recommended as a moderately easy project for further work."

Section 4.1 ended with a discussion of the novel of concept "**hypercolumn connectionism**" (as opposed to "**neural connectionism**") which was instrumental in

laying the foundations of our eventual subhypercube machine. The getagroup algorithm (section 4.1.1.6) is an implementation of the cr theory in section 3.4.4.1. The difficulties of the getagroup implementation encouraged us to search for a more useful theory than the powerful cr theory. This search resulted in a top-down perspective which led to the subhypercube theory of section 3.4.5.3. Thus the hypercube representation is an attempted abstraction of the hypercolumn (section 3.2.6). Attention then shifted to the realisation of the analogous hypercolumn components as the shape of the solution set within the hypercube. The subhypercube theory naturally leads to a convincing abstraction of the hypercolumn components as 'polytopes' or subhypercube groupings (section 3.4.4.1).

4.7.1.2 Summarising of section 4.2

The search direction was changed from the above to a search through the subhypercube heterarchy both from the most general and the most specific ends and thereby coalescing upon the solution in between. The basic learning algorithm was refined from an initial "global0" through global0.1, global0.2, ... to global0.9 which was later renamed "**Learn**". We indicated the scope of the experiments as approximately 30,000 runs undertaken. The work described was from a representative class mix out of this range and typically the experiments presented were those which first indicated the various effects. We needed to have a **standard** by which to compare these algorithms. We chose **parity** since it is an excellent and well accepted benchmark for testing connectionist networks. As a defined problem it is: independent of the dimensionality or domain, the hardest possible nonlinear problem within the dimension, it also serves as a datum by which to compare and hone refinements to the algorithm. We have experimented with parity as far as the 15th dimension. We believe that this is in itself easily a record for such systems. The H^{15} parity problem takes the final algorithm nearly a day to learn.

The exponential explosion of the algorithms was discussed and illustrated graphically. We noted the usefulness of asking the basic question that we are concerned with namely "What is the nature of this curve?" In effect: "Which points are the most influential?" A reoccurring process in developing a faster and faster subhypercubing algorithm has been to continually ask this question and find by various means where any inefficiency still exists. Such methods and their resulting insights greatly helped to refine the algorithm during its developmental stages. For example the method of **multipliers** was found to be a particularly useful indicator of the efficiency and quality of the algorithm.

The well known "**rule of four**" as in Tesauro & Janssens (1988) has been postulated as the theoretical limit to work in neural nets. However we theoretically deduced that the *subhypercubes* multiplying factor is 2k/p; a simple and useful expression which applies in the general case. This resulted in a multiplying factor of 3^D in contrast with 4^D.

Several other factors intervene to reduce the upper limit of the 3^D factor even further. How far is a function of certain assumptions made. This problem dependency complicates the theory. There is no easy relationship between the multiplying factor and the subhypercube multiplying factor only an upper bounded relationship. Practical work likewise revealed that the relationship is quite complex. The more consistent results obtained for the final algorithm 'learn' merely reiterated these and other results although in greater detail. The resulting curve shows that the individual multipliers vary from about 2 to 4 over the lower dimension range. The mean is about 3 while the graph becomes asymptotic to about 4.25.

We discussed the development of the algorithm from the perspective of its basic concepts. One useful facility is the underscore "_" in the Index list to indicate a variable or ϕ-boolean. One advantage of this notation is that it is possible to input *variables* in the exemplars. This is highly useful in a number of respects. For instance, we are then able to say in effect: " ... all examples of this type act in the following manner ..." This enormously reduces the number of examples that may otherwise be required to adequately define the space; leads to a very compact formulation of the result by the orthogonality of representation; and is a powerful facility which is *not* available, for example, in conventional neural nets.

Various problems and difficulties in the development of the algorithm were discussed. For example, list manipulation being three times faster than database access which gave a confusing initial advantage to early algorithms; the significance of database redundancy, duplication, etc.; environmental effects causing the first two runs to be slower; the difficulties of **uniqueness**; typical, worst and best case problems; dynamic database storage of subhypercubes; consequent database access speed loss; and so forth. The core of the algorithm is the **doublebeamsearch stack cycling machine** which was illustrated and some of its subtleties discussed in detail.

Difficulties with QUINTUS PROLOG in connection with recursion were problematic. But so, we stated, was our solution **iteration** - a "software engineer's nightmare" and "The marvellous benefits and beauty of Prolog are precisely balanced by its unacceptable shortcomings and enforced ugliness".

In order that the practical algorithm abide by our theoretical insights we were forced to employ some assumptions regarding the circumstances when the subhypercube is orthogonally assumed to be "**complete**". Sparsity means that some connections may still unknown. Our assumption is that if there is *no evidence against* the subhypercube being "complete" we assume it to be *complete as far as is known*. This led to an implementation of our sought after simplest possible mathematical explanation. Similarly, if there is no new positive exemplar information *not already taken account of* to suggest otherwise we assume the subhypercube to be negative, otherwise we will invite duplication of effort at some lower

level. These subtleties were explained and illustrated. We also stated that it is *most efficient* to test for only **one** such pair, *whatever the size of the subhypercube,* in order to ensure the requisite necessity for expansion. This neatly fights the exponential explosion. It became one of our major techniques. Indeed to capitalise on this advantage we attempted to reapply the same approach throughout the algorithm - but with only partial success. We further discussed correlation of the practical algorithm with the theoretical unbiased situation, the resolution being based upon a "weight of evidence" argument for assumptions.

We assume that, in order to reflect the universe, the data has a reasonable spread across the universe (Huyser & Horowitz, 1988). From this we noted that the subhypercube algorithm in the 'non-learning' case usefully acts as an **information reducer** and for the alternative 'learning' case we have **resolution**.

With regard to the design of the code segments many points of efficiency, technique, etc. were raised. For example, one big advantage was **unification** such that matching then experimentally realising a 50% speedup. Another advantage is that unlike most machine learning and connectionist algorithms where the learning time cannot be calculated prior to learning, the system is able to output estimates as guide-lines to the user on the maximum probable learning time by taking into account the particular parameter set used in the problem.

Further work in this area should even enable a reasonable estimate of the learning time for *any* problem. This is possible by comparing the parity distance of exemplars via their percentage linearity/non-linearity calculus. A firm basis for efficiency gains was often found to lie in the structure and rudimentary intelligence of the algorithmic environment as evidenced by the init_learn procedures.

A number of further points with regard to the final algorithm were discussed: such as the 'makesupervars' solution to the duplicate subhypercubes problem; **depth resolution**; **noise** by analogous "shrink and dilate principles"; and the very marked speedups due to the **Top_bot** and **Binary chop** techniques at no loss of accuracy for Top_bot and proportional to the non-linearity for Binary chop.

Binary chop considers *all* nodes in the universe, but it ignores all "bridging subhypercubes" between the major opposing pair of subhypercubes which is being considered at each level. This is a very useful side-effect since, for example, the complete solution is still found for the most non-linear of problems, i.e. parity; whereas highly linear problems are tractable anyway, for example by Top_bot. This leaves intermediate complexity problems to be most easily tackled by Resolution. It would be useful further work to see if pre-inspection of the exemplars (by estimating the complexity) would allow the automatic choice of the best strategy. The format of the resulting formulae is:

formula(*Indicator, [Class | Subhypercube]*),
where the 'Indicator' takes the value 0, 1 or 2 in order to differentiate between a classification of true, false or incompletely processed subhypercubes (as resulting from resolution) respectively.

The major experimental results from this developmental phase are as follows. Parity in the 10th dimension by "binary chop" takes learn only 73 seconds which is very fast. Making the procedure makesupervars failure driven improved the time from 1 day to a mere 95 seconds. This remarkably positive result is at least partly due to the way QUINTUS PROLOG works (because of the elimination/reduction in garbage collection) regardless of the advantages of the algorithmic subtleties of iterative prolog. At lower dimensions the effect is much less marked. Rewriting the 3 concatenation makesupervars as single concatenation machine speeds up the algorithm by only 1 second at the 10th dimension. Making h(I,C) as unified list as in h([C|I]) for faster indexing, referencing, unification, etc. speeds up the algorithm from 95 to 73 seconds at the 10th dimension.

Other changes had less dramatic effects. Often the advances are difficult to illustrate graphically and the gains only minor but cumulative. In contrast, the elimination of recursion had sufficient effect on efficiency to be clearly seen even in the logarithmic graph where we note that the speed-up is very considerable where gains are hardest won, that is, at the higher dimensions.

4.7.1.3 Summarising section 4.3

As we stated in section 3.1 the **purpose** of experiment lies in "**testing the predictions of the theory**". This we did in accordance with our scientific cycle outlined in section 3.1.3. In section 3.4.5.4 we deduced that the: "theoretical predictions are precisely the same as the 11 requirements" which therefore "would form an excellent series of practical experiments due to their solid links into the theory (section 3.1.3)". Our 11 requirements were: simplest, global, balanced, correspondence, targeted, faster, virtual, actual, ϕ-boolean, associative and resolution.

After experimenting we report that: "we have been **unable** to demonstrate experimentally that the machine does **not** act in accordance with the predictions from the theory". This is precisely aimed: experimentation with a defined purpose. We remember that experiments cannot, of course, prove the theory (section 3.1.2). We merely, at best, retain the theory pending more exacting experimental demands. Our conclusions regarding the testing of the theoretical predictions may therefore prove to be ephemeral. Yet this was an **important set of experiments** which had been carefully thought out. We also discussed our *general* approach to experiments including various experimental wrinkles.

Further important points brought out in this section regarding the design and operation of the machine were as follows:

1 Strong preference for : "the **simplest** possible **globally** derived explanation that fits the data in a **balanced** way with respect to both positive and negative evidence."
2 Implements our **MEU theory.**
3 A deliberately **anthropomorphic** machine.
4 Uses a novel approach to handling complexity in learning which we call **metrical.**

4.7.1.4 Summarising section 4.4

We stated that it was a second major exercise demonstrating the practical applicability of the theory by addressing the scope and limitations of the machine. Considering scope, we discussed an experiment using complete convergence with a representation of mRNA codons and concluded our machine to be useful as a tool for inspection purposes or for designing fail-safe codes. Also similar but much more complex problems in genetics and other domains could perhaps be usefully tackled.

Secondly we compared our machine with a well known genetic learning system called "BOOLE" by Wilson (1987) and an ID3-like decision tree classifier called "C4" by Quinlan (1988). The learning task concerned was in the multiplexer family. The subhypercube was significantly faster on smaller tasks and was more easily able to reach complete convergence.

Thirdly simple experiments in the domains of natural language (morphology, sentence structure and word prediction) and a basic user interface message system encouraged us to conclude that the hypercube is quite well suited to tasks definable as a finite state machine. The resulting high information compaction could be especially useful in real time work. We suggested that object oriented design may be well suited to our approach. One problem was the need for examples of invalid states - which in some cases may not be available - it would be preferable to have available multiple output classes to define the next state. Further experiments concerned the logic of distinctions, fixed and variable shapes considering change of size, shape, translation, rotation and the same ideas in a more abstract way using logical implications between opposing pairs of concepts within a network. Work was also conducted to build up a simple symptom disease representation in the field of urinary tract infections.

We comparing our metrical machine against the statistically biased ID3 machine. We illustrated how the ID3 method offers no help in avoiding a poor choice of attribute to split on. In contrast our method does "see" the more relevant attributes because it is a global

searching algorithm and makes metrically-based decisions with a bias towards the simplest possible balanced solution. The solution found was arguable from the data, closely predictive of the global minimum and also simple.

Pattern experiments concerned the intelligence test type of problem such as the train problem and usually seemed to present the machine with little difficulty. Patterns in the form of simple shapes were another useful source of experiments, as in simulating actual hypercolumn patterns at the same resolution. Simpler patterns were also used to test the machine's tolerance to noise. Our approach to noise involves a metrical "erode and dilate". Noise tolerance seems to be robust and reliable considering the simple technique used.

As we discussed in chapter 1 we originally decided to attempt to theoretically define a coalesced machine in order to combine features from several fields of machine learning. In retrospect we have made some progress, our machine has some seminal, useful and novel features. Its limitations mostly stem from decisions made much earlier on, typically in the earliest stages of the theoretical work. Absent, but useful features include multiple classes (hence also encoders and unsupervised learning), larger grain-sized representations, feedback, competitive learning, spontaneous recovery, real-values and a conventional structure of hidden layers. Attempts to provide these features as an afterthought do not lead to satisfactory solutions. However, we strongly urged that in finding ways to extend the theory to include some of these characteristics we must also retain our present hard won gains and crucially important insights into the nature of learning.

4.7.1.5 Summary of section 4.5

Our global subhypercubing learning machine has specific advantages when compared with other learning machines and in particular connectionist machines. The first of these advantages relates to its consequent reliability, dependability and reproducibility. In short the machine has a **predictable behaviour**. Same problem, same learning time, it never "gets lost". This covers an aspect of what we mean by the term "metric". We illustrated this "**metrical**" progression of the algorithm by a very simple worked example.

The advantage of depth resolution is that it sets a limit on the search depth and is therefore used when the complete solution is not required. A disadvantage is that its usefulness is problem dependent, but since resolution allows much faster processing it provides a significant novel contribution such that an approximate solution may be produced in an acceptable learning time even for large problems. Predictability allows guidelines to the user on what is the maximum resolution depth that would lead to an acceptable solution within a very short time period (a feature unique to this machine). We illustrated resolution with an example and followed that with a review of learning curves.

Two other novel features allowing significant algorithmic speed-up are "Top bot" and "Binary chop".The parameter "Top bot" refers to a test of the top and bottom node of each subhypercube. Since any given node is connected to half of the complete subhypercube this allows an opportunity to speed up the search considerably. Another parametric was Binary chop which takes advantage of the fact that all nodes in the subhypercube appear in any two opposing hyperplanes. Binary chop can only be used to obtain approximate solutions but again allows a very significant speed-up. A further experiment tested for but was unable to find further 2nd order combinatorial effects using binary chop and learn.

Finally we provided machine comparisons with Hypersphere Classifiers, the conventional machine learning cycle and several specifically optimised Error Back Propagation learning machines. In the latter the subhypercube machine was seen to be not only significantly faster but even more so with increasing dimensionality. We also listed some heuristics learnt as a result of our project.

4.7.1.6 Summarising section 4.6

The fifth requirement (section 4.1 and section 3.1) as the minimal necessary work to obtain the requisite credibility for our work concerned the need for "An advanced application of the machine". We have already referred to the dangers inherent in impressive demonstrations of this kind as "hype" which, together with its often attendant lack of a defined scientific approach, is **our nemesis**. We therefore proceeded and have reported with **extreme caution.**

A joint venture between the University of Florida and Aston University involves a novel approach to the prediction of the partition coefficient of compounds. We report four **significant** initial findings upon using the hypercube for this research, namely: the hypercube can cope with the problem size at the **68th dimension**, learning times are **fast**, the problem was **real world** and the problem was **totally unseen** yet worked first time. These results are however very preliminary since obtaining more detailed results from this project would certainly occupy several years of research work.

4.7.2 Discussion.

We offer a few remaining general comments herein which had no place elsewhere.

1 We do not believe that any panacea will be found to the problems of machine intelligence. Rather each useful machine has its own distinct advantages, disadvantages and special uses as in Minsky's "Society of Mind" concept (Minsky, 1986). These features should not be exaggerated. A novel theory should not be shot down merely because it does

not do "X". There will always be limitations of any machine. Rather, a balanced viewpoint should be taken. Analogously, it would be ridiculous to complain that hammers should be discarded because they are poor at sawing. Hammers should be used as hammers, at which they excel, and saws should be used as saws, at which they excel. Machine intelligence is similarly accumulating a varied "toolbox" where each "tool" has its place. On the basis of work undertaken on this project we recommend that the subhypercube algorithm be considered as a useful complement to the existing components of the toolbox.

2 In this regard what kind of a tool and for what purpose is the subhypercube algorithm? The subhypercubing algorithm has been designed as an experimental basic "building brick" (section 3.1) to test out the theoretical ideas of chapter 3.

3 In particular our machine has its main virtues as a reliable "low-level," non-symbolic, domain independent learning machine and only with cautious use for higher level symbolic learning (as sections 3.5 and ensuing sections) with output to an expert system. The unique advantage of our machine is that it is *"mathematically analysable in depth"* (section 3.4.4.2). This advantage followed naturally from a thorough analysis of requirements, a solid understanding of the problem, the orthogonality of the machine itself and a rigorous elimination of the ad hoc.

4 Unquestionably, despite the practical work, **it is the theory that is the highlight of this thesis.** A theory has been outlined, which, starting purely from set theory and building a mathematical basis in lattice theory, ends in a specification for a machine which can inductively learn by example. In contrast the *programming* of A.I. offers no solution due to the complexity of real world patterns to be encoded which require our brain for an adequate solution. As Whitfield (1984) suggests, the pattern number (dimensionality) of such problems is of the order of the number of elementary particles in the known universe or possibly greater. [1] As Neumann (1958) says the real problem is, indeed, how to handle **complexity**. This, most basic problem, we have attempted to address in small measure in this project.

4 In section 2.1.2.5 we referred to the work of Kodratoff & Ganascia (1986) on improving the generalisation step. We recall that their method involved two basic steps: detection of structural matchings ("difficult") and the generalisation phase proper ("easy"). In our hypercube approach, the difficult first step is greatly facilitated by the metric

---

1 In the brain such tremendous complexity is easily handled. This appears to be due to the number of synapses per neuron (20,000 and 10^{10} respectively), the number of neurotransmitters per synapse (18 at least) affecting each synapse, the number of neurochemicals sometimes affecting the "chemistry of the wiring" (estimated to be of the order of 100,000), the number of ways synapses can form between axons, cell bodies etc. (estimated at 9, at least, and ignoring the more complex spline, button, etc. structures) and the number of variations of the synaptic size (estimated to be 18) possible during growth. This still does not account for the further complexity of the dynamic, analog and statistical nature of neurons, on top of the purely digital aspects (Neumann, 1958), of which there are as yet no estimates.

orthogonality of the lattice structure. Further, this eliminates any need for heuristics with their associated domain specific dangers. Our detection of structural matchings is general to specific in direction. This, in turn for the hypercube, leads naturally to Kodratoff & Ganascia's second step, detecting common variable bindings as concept subsumption, which therefore, for the hypercube, is specific to general in direction. These two steps mesh and iterate well. The boundaries on the solution thus converge from both directions to the final solution or solutions. Unlike the version space method, both disjunctives and/or noise are allowable, yet the system still "knows" when it has finished. By-passing is impossible.

4.7.3 Further work.

In section 4.1.1.2 we suggested that: "Simple comparisons between QUINTUS and C suggest a possible speed-up by recoding of up to 10^3 times. This could turn the work into a valuable fast product and is to be recommended as a project for further work."

We noted in section 4.2.4.4 that intermediate complexity problems are most easily tackled by Resolution. It would be useful further work to see if pre-inspection of the exemplars (by estimating the complexity) would allow the automatic choice of the best strategy. How can this be approached? Section 4.5.1 hinted at automatically predetecting parity. This looks possible, but it is not very useful. Generalised to detecting the average linearity level, it could be very useful. If the average complexity is known then a "level to cube" transformation if iteratively repeated will raise the cr close to linearity! This is another way of saying any nonlinear function can be completely linearised by raising its dimensionality. The transformation is **trivial** and remarkably powerful. The trick is knowing **when** to do it, hence our "pre-inspection" comments above.

Section 4.4.3 on the limitations of the machine noted certain absent, but useful features which included multiple classes (hence also encoders and unsupervised learning), larger grain-sized representations, feedback, competitive learning, spontaneous recovery, real-values and a conventional structure of hidden layers. Attempts to provide these features as an afterthought do not lead to satisfactory solutions. However, we strongly urged that in finding ways to extend the theory to include some of these characteristics we must also retain our present hard won gains and crucially important insights into the nature of learning.

There is a very pressing need for connectionist machines to be able to operate *dynamically*. Whitfield (1984) and Neumann (1958) both draw attention to the importance of dynamics in neural networks. Very little work has been done in this area - yet the static systems extant could certainly not represent the coupled oscillations, "burstar feedforward" "Are you there?" type of circuits, "ringing circuits" etc. found in biological systems of even the simplest kind. Adding "time" very considerably adds to the complexity of the system.

Without it, the loss may be catastrophic to real-time simulations. A possible starting point may be to consider what exactly is the advantage of a dynamic system over a static system. Since a computational explosion results with all possible variations to be accounted for, the hypercube could still be of value by encoding zeroth, first, second, ... order effects (x, dx, d^2x, ...) either by booleans or a boolean encapsulation of a higher resolution. Such a simple approach to the dynamics of the problem could well capture problems beyond the scope of differential equation analysis as a first order resolution of the problem.

Expanding the scope of the system by considering hierarchies and networks of hypercubes is enticing.

We are interested in a Tensor reformulation of our work since we feel that it would offer the advantages of easier extrapolation and extension of the ideas presented herein. Caianiello is the best known and most prolific source of work in this area (e.g. Caianiello, 1988).

Maintenance has not been addressed in this project yet it is a very necessary part of real world systems. For example, consideration and further work needs to be done in the areas of: replacing outdated and incorrect knowledge, integrating new knowledge, true incremental learning etc. This work could be done outside by front-end machines but more fundamental difficulties may require changes within the hypercube machine.

Topological changes to the hypercube might be considered or using the hypercube approach on a different topology is also possible. The topologies possible for connectionist machines are many. For example: grids, trees, hypercubes, hypertoruses, omega networks, indirect binary n-cubes, banyan networks, twisted toruses, k-folded toruses, x-trees, shuffle exchanges, k-way shuffles, crossbars, Batcher networks, Clos networks, De Brujin networks, Delta networks, hashnets, reverse exchanges, butterfly networks, spin models etc. and combinations of all these are also possible, as in Youssef and Narahari (1990). Hillis considers some such groupings (Hillis, 1987).

Multiple hypercube architectures are seductive due to their promise of greater power. Examples of possible multiple hypercube architectures are: "hyperhypercubes" and "hidden layer hypercube machines". "Hyperhypercubes" implies that there is an n-cube hypercube at each vertex of the 2^n hypercube network. "Hidden layer hypercube machines" implies that the output of each layer forms the input to the next only the input and output layer being specified. The hidden layers are pre-specified by the hypercube dimensionality of the positive outputs, facilitating coarse coding, but this is only available at runtime.

A feature common to these studies is the three-state variable representation used. But since the non-boolean state is only resolvable into the two-state, the system can only

ultimately store and retrieve binary patterns or variables or functions of such. Two longer term requirements are: firstly, graded response variables allowing the exploration of more general activity patterns and secondly, continuously variable firing rates allowing true dynamic behaviour.

This could well imply a statistical mechanics flavour to the analysis within a global framework rather than the tertiary (collapsing to binary) spin approach taken herein, as for example, if we considered an analogy with the modulated spinning helical vector of magnetism. Perturbations of this kind could then allow the sought after "real" dynamics. [2] Or if our spinor was decimal based then we have 0 to 9 quantization. That is a "decimal hypercube". The database would then be say: h([6,4,9,0, ...]). For example, considering the boolean hypercube from 1 to n translated to the non-boolean hypercube of m discriminants we have:

$[a1,a2, ...,an]$

translates to:

$[a11,a12, ... ,a1m, a21,a22, ... ,a2m, ... ,an1, an2, ... ,anm]$

4.7.4 Conclusions.

First, we consider performance of our machine to other learning machines of note. Second, we review our scientific cycle. Finally, upon discussion of the unique advantages of our machine we suggest our preferred way forward.

4.7.4.1 What have we learnt?

The **best algorithm** depends upon the answer to a number of questions (most of which we have discussed herein) such as: how linear or non-linear is the problem; how high or low is the problem's dimensionality; what is the level of noise; is there time to do hundreds of trials to find the fastest learning time; is reliable, predictable performance the major criterion; is a completely accurate solution is required; is an approximate solution sufficient; are multiple output classes required; must it handle symbolic information; how flexible should the representation be; is it required to interface with conventional systems; should its output be runnable on an expert system; should it be able to cope with the severest of problems; does huge variability in its learning times for the same problem matter; is high speed the major requirement; are the problems of the target domain recurrently similar or

---

2 Processing power would need to increase for the benefit to be felt. But remember that the first layer of each human retina alone performs $3.5 * 10^{10}$ binary descriminations per second on an input field of thousands and as we saw von Neumann observe: it then performs a binary to dynamic translation on top and actually other jobs as well such as comparisons with surrounding cells, averaging the results, detecting light to a resolution of *one* photon, primary colour coding etc! We need a *lot* more power!

highly variable; does it matter if the algorithm does not converge periodically; is the fastest time required; or is an average time the most critical dimension; and so forth.

Clearly, there is **no overall winner** amongst the various learning machines. There is no panacea. It all depends upon the particular answers to the above questions. In some situations, often involving the hardest problems, Genetic algorithms such as Boole will be preferred, but generally they are chronically slow. In other cases Neural net algorithms such as Back Propagation and its variants are superior, particularly where there is a lot of noise, the dimensionality is high and the complexity is not too non-linear. Studies quoted herein and elsewhere suggest that Machine Learning algorithms such as ID3 and its enhancements are often very much faster than neural nets for intermediate level problems. The hypercube also seems to find its niche. For problems of low dimensionality, even for highly nonlinear problems, as such, and for moderate noise levels it is very fast and extremely reliable, predictable and accurate.

Yet comparing algorithms of such complexity is difficult. Enhanced versions of some algorithm, which then overcome prior objections, often become available before the results of such studies are known - thereby probably invalidating the results. Comparisons are also fraught due to the lack of sufficiently detailed reporting in the literature of precise times, hardware and software used, etc. For example, neural net studies often quote "epochs", yet how many seconds of processing time on what machine under what conditions for what problems etc. does this correspond to? Tracking down the answer is we have found a time consuming occupation.

In assessing the worth of any new research serious comparisons with existing work are mandatory. Failure to compare means the loss of the chance to understand what it is about one algorithm that makes it better or worse than another. Ultimately such failure leads to independent lines of inquiry and duplication of effort - a much observed phenomenon. Yet, as we have just illustrated, the most painstaking comparative study is doomed to achieve only a fuzzy answer at best. We have done our best to compare the hypercube with three leading machine learning algorithms along the most critical dimensions. A useful diagram is said to be worth a thousand words. We therefore compactly compare our Subhypercube algorithm referred to as "Hcube" against three other groups of leading and well respected algorithms which we shall refer to as: "BP+", "ID3+" and "Boole". See Figure 4.68.

In the diagram BP+ refers to Rumelhart's "Back Propagation" and its variants, which is easily the best known and most widely used Neural net algorithm. (In section 4.5.5 we compared the subhypercube machine with fast versions of Back Propagation where the sigmoid tends to a hard limiter). ID3+ refers to "Interactive Dichotomiser 3" by Quinlan and its more recent enhancements ID4, ID5, C4 etc., which is the best known Machine Learning algorithm. (Again in section 4.4.1.2 we compared a recent fast decision tree machine, "C4",

with our subhypercube algorithm). Boole is possibly the best known Genetic algorithm at present (compared in section 4.4.1.2).

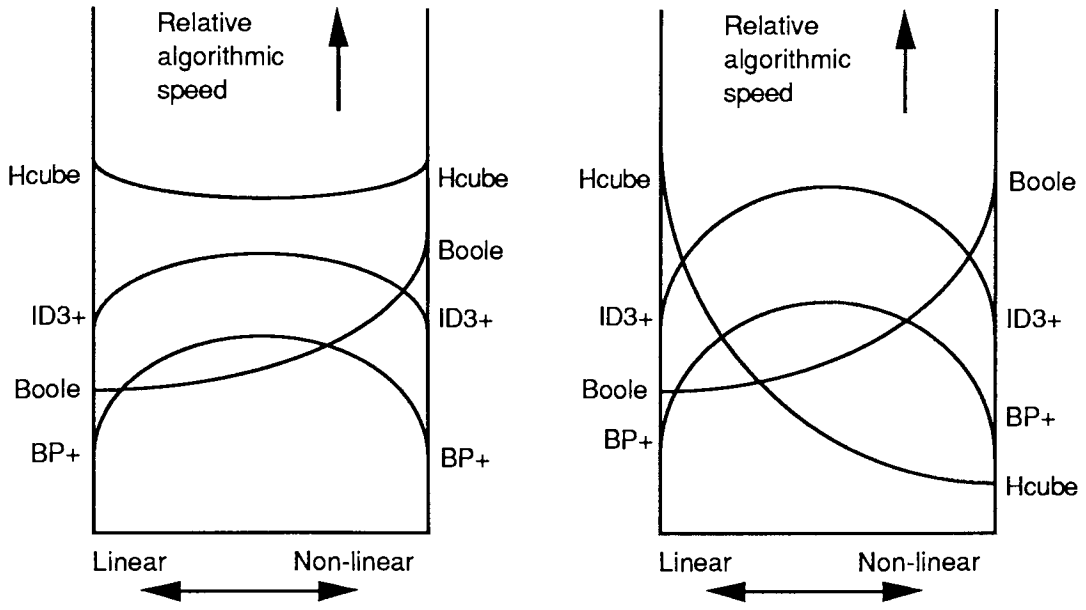
Let us make very clear that the source of these comparisons is varied - some of it being work we have done and reported herein, some of it is the work of others reported in papers, some of which are quoted herein. For example, we have compared the hypercube with C4 (type: ID3 decision tree) and Boole (a Genetic algorithm) using the *same* machine (Sun 3/50) both running under the *same* Sun unix operating system, (but using different languages) on the *same* problem, as we have already seen in section 4.4.1.2.

Also we have carried out a number of comparisons against various versions of Back Propagation again on the same machine (SPARCstation1) both running under unix (but using different languages) on the same parity problems (section 4.5.5). Also several studies quoted herein have compared BP and ID3 etc.

As a result let us also make very clear that, even so, because of the difficulties of comparison the conclusions are a relative measure only and must ultimately be regarded as partly intuitive. Further, once again in accord with our comments in section 3.1 we caution the reader that: *it is risky to generalise on the basis of a few comparative studies of isolated problems*. Further caution ought to be issued. The various algorithms each have unique facilities or facilities not common to the group as a whole. We have only compared that which can be compared. So we may not have used all, or even the best, facilities of those algorithms in the comparison.

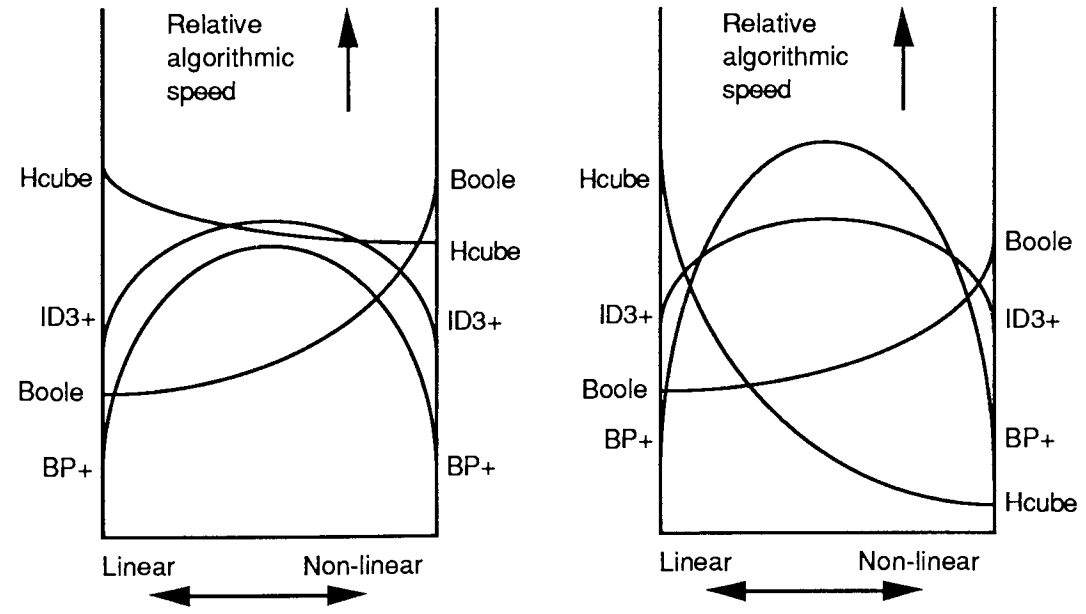
This puts a different perspective on any unfavourable comparisons along some particular dimension and complicates the picture. For example, if algorithm A is two orders of magnitude faster than algorithm B this fact may be irrelevant if what is required for a particular application is facility X which A cannot supply but B can. That all understood, we nevertheless have some reasonable degree of confidence in these findings, within the above terms of reference. Our comparison, fuzzy in conclusion for all the reasons given above, is we believe realistically summarised in Figure 4.68.

Figure 4.68 suggests that **the subhypercube algorithm has value and is worthy of further research**. Most importantly there is an opportunity to express what we feel to be the main strengths and weaknesses of the algorithm as a result of our research and these comparisons. The main points of interest which are apparent from the "hcube" curves are: excellent performance for low dimensions, degrading a little under high levels of noise but still good; and high performance, independent of dimension or noise, for fairly linear problems although degrading rapidly thereafter. Clearly as the dimensionality increases the machine becomes of less value due to the toll being taken of by the exponential explosion.



(a) Low Dimension, low noise.

(b) High Dimension, low noise.



(c) Low Dimension, high noise.

(d) High Dimension, high noise.

Figure 4.68 Relative worth, comparison of learning algorithms.

4.7.4.2 The hypercube cycle for machine learning.

We stated in section 3.1 that one *raison d'etre* of this project was to assess the applicability of a truly scientific approach to non-symbolic learning as a necessary step towards limiting the number of "hidden variables" involved; and we feel our success in this endeavour to be encouraging. The source of our scientific approach (sections 3.1 to 3.1.3) being partly experience, partly observations on the success of the work of others and partly the writings of Mach, Kant, and most particularly Popper who has lucidly analysed the thoughts of many others (Popper, 1974).

In section 3.1.3 we promised that we would complete that section in the concluding part of the thesis. We now do so. For the hypercube, the loop of four states in the scientific cycle translates as follows.

1) The psychological experiments lead to our assumed fundamental fact that in the absence of disturbing factors the brain prefers the simplest possible induction consistent with the evidence.
2) The simplest possible theoretical machine which works in this way leads deductively, in the general case, to:
the theoretical subhypercube machine predictions.

3) A program was written to work according to the subhypercube theory and the theoretical predictions tested on it.
4) Pivotal aspects of this theory or its inverse were then assumed *incorrect* as a basis for experiments. These experiments after analysis led to:
the experimental result,
that it is incorrect to assume that the subhypercube theory is incorrect. Further experiments considered the scope, limitations and application of the practical implementation of the theoretical machine.

But this does *not* mean that the subhypercube theory is correct or even useful; merely that the theory should be retained pending further experiments, rather than discarded, as in the case of it being disproved. Hence the final seemingly weak conclusion of the project is that this work is worthy of further research. This is the correct scientific stance in the circumstances.

4.7.4.3 Looking towards the future.

It is worth while saying that: ***the particular combination of advantages of our theory would have been very difficult to discover by any other means.*** Many aspects of this thesis are extremely interesting, even instructive. Aspects such as:

1 The fact that we have shown that **weightless learning is possible**. We attempted to prove that weights were necessity by first eliminating them, only to find that weights are **not** essential to connectionist learning - this was a surprise. And even more so was their subsumption by polytopes,

2 The **metrical** nature of the search process. A crucially important comment was that: "The subhypercubing algorithm is theoretically **deterministic** for finite hypercubes with defined start and end points in its convergence and cannot, therefore, "get lost" or enter an infinite loop" (section 4.1.1.2).

3 The ability to reliably and monotonically reach **complete convergence** even in the presence of moderate levels of noise,

4 The **global nature of the search** thereby avoiding the local algorithm's nemesis of ravines,

5 The ability to handle **disjuncts** including mutual exclusion as well as the easier conjuncts,

6 The ability to invariably produce **minimum polynomials**.

7 The fact that we dared to challenge the "obvious" assumption that it is of no interest to consider all possible solutions because of the exponential explosion led to **novel ways of reducing this exponential explosion**,

8 The value of an integrated approach - a **coalesced machine** (section 1.3), which does **not** just ignore all the excellent prior A.I. work in representation, Machine learning etc.,

9 The fact that at least for low dimensions all the criteria of section 1.2 came to some reasonable fruition: input felicity, learning speed, noise immunity, accessibility, symbolic or connectionist in operation, independent of the knowledge level, fully automated and therefore hidden well away from the user and with a unifying representation of very extensible power, namely boolean,

and so forth ... led to very definite advantages in lower dimensional spaces. That said, it is still predictably true to say that the combinatorial explosion also means that these very desirable characteristics nevertheless lose power as the dimensionality remorselessly increases. This point was **so obvious** that no one attempted to see if, nevertheless, there might be things to be learnt by so doing. But it is also true to say that *anything so discovered may well be both novel and insightful.* On the other hand we do need and would find a considerable increase in possible applications if the machine also possessed, for example: multiple outputs, real valued processing, hidden layers and the ability to cope with high dimensions.

We ought to think very hard indeed about how to improve the algorithm without watering the whole down so that we reach a low point whereupon the power and value of our

present research is diminished and any additional supporting features imported from either machine learning or connectionism are equally constrained in effect. To do so would be to fall between two stools. The real power of this research is in its pointers towards **novel directions for future work**. And that takes a big effort to work out **exactly** what it is that is doing the trick or its converse. It is hard because it is a credit assignment/blame problem and in a complex system such an answer can escape even the dedicated. It appears that it is necessary to make tradeoffs, but we have become attached to our theory's strong points: appearing as they do to provide a new theory and insight into the very essence of learning - in strong contrast with heuristic and pragmatic approaches. Yet there may be a way out ...

One possible pointer towards building upon the present weightless theory to obtain a **metrical real number space** with the **best of both worlds** is to be found in the 1957 work of the soviet mathematician Kolmogorov. In a remarkable paper Kolmogorov states a theorem concerning the mapping of continuous arbitrary functions of the unit hypercube (implying convexity) into real numbers in terms of purely one dimensional functions, thereby neatly solving Hilbert's 13th problem. The theorem was later refined by others (Sprecher 1965, Lorentz 1976). This work suggests that the weightless mapping for lower dimension D could be implemented by a three-layer net of D (input), 2D + 1 (hidden) and K (output) nodes per layer. We hypothesise a more powerful simulated hypercolumn possibility than at present by dividing the higher dimensional integer interval 0 to x into x equal parts each of unit interval (0 to 1) and processed as above in order to produce an algorithm which is a fast global metrically indexed set of continuous real monotonically increasing functions of arbitrary resolution f(x). Each sub-mapping would then seem to correspond to a Minkowski - r power metric of "city block" error metric equal to at most 1. Further, if preprocessed cr sampling indicated significant nonlinearity, a level to cube transform would significantly increase the linearity and therefore the learning speed of the whole even faster.

CHAPTER 5.

Conclusions.

5.1 INTRODUCTION.

Two criteria apply with regard to our concluding chapter. Firstly, no new information is presented in this chapter. Rather, it merely summarises all previous summaries. Secondly, we take the view that the conclusion (like abstracts and introductions) should be short and to the point. If it were not so the conclusion itself might be in need of further chapters to summarise, introduce and conclude it!

This short chapter provides a precis and conclusion to the thesis, under four headings. First, we provide a summary of the chapters. Second, we discuss the major achievements, advantages, disadvantages and limitations of the work. Thirdly we outline suggestions for further work. Fourthly, we offer our conclusions on the project as a whole.

5.1.1 Summary.

This thesis describes a novel connectionist machine utilizing induction by a Hilbert hypercube representation. This representation offers a number of distinct advantages which are described. We construct a theoretical and practical learning machine which lies in an area of overlap between the three disciplines - neural nets, machine learning and knowledge acquisition. Hence we refer to it as a "coalesced" machine. To this unifying aspect is added the various advantages of its orthogonal lattice structure as against less structured nets. We discuss the case for such a fundamental and low level empirical learning tool and the assumptions behind the machine are clearly outlined.

One assumption is paramount. Unnecessary proliferation of theories should be much frowned upon in the absence of a watertight first cause. Hence one purpose of chapters 1 and 2 was to provide just such a justification. Chapter 1 provides an historical overview and brief introduction to the subject. Chapter 2 provides a thorough review of past work in the field of learning in general and of those learning machines of particular interest for this project. Further, in order to provide the reader with the necessary equipment required of later chapters we endeavoured to analyse the past work in sufficient depth to fit the reader with some intuitive feel for the subject.

Chapter 3 began by outlining a "scientific methodology of hypercube learning" as set of guidelines and a timetable to be strictly observed throughout the rest of the project in the theoretical work undertaken and in the practical work (both experimental and

developmental). We then introduced the basic concepts involved: an orthogonal lattice structure the Hilbert hypercube of an n-dimensional space using a complemented distributed lattice as a basis for supervised learning. Psychological experiments resulted in our "unreasonable theory" of how a machine might learn using the hypercube representation.

The resulting "subhypercube theory" was implemented in chapter 4 in a development machine which was then used to test the theoretical predictions again under strict scientific guidelines. The scope, advantages and limitations of this machine were tested in a series of experiments.

5.1.3 Major achievements, advantages and limitations.

Novel and seminal properties of the machine include: the "metrical", deterministic and global nature of its search; complete convergence invariably producing minimum polynomial solutions for both disjuncts and conjuncts even with moderate levels of noise present; a learning engine which is mathematically analysable in depth based upon the "complexity range" of the function concerned; a strong bias towards the simplest possible globally (rather than locally) derived "balanced" explanation of the data; the ability to cope with variables in the network; and new ways of reducing the exponential explosion. Performance issues were addressed and comparative studies with other learning machines indicates that our novel approach has definite value and should be further researched.

Limitations of the machine include certain useful features which may exist in other machines such as: multiple classes (hence also encoders and unsupervised learning), larger grain-sized representations, feedback, competitive learning, spontaneous recovery, real-values and a conventional structure of hidden layers. Attempts to provide these features as an afterthought do not lead to satisfactory solutions. However, we strongly urge that in finding ways to extend the theory to include some of these characteristics we must also retain our present hard won gains and crucially important insights into the nature of learning.

5.1.3 Further work.

We suggest the following possibilities: recoding in C as a much faster, portable language; pre-inspection of the exemplars (by estimating the complexity) would allow the automatic choice of the best strategy; expanding the scope of the system by considering hierarchies and networks of hypercubes; tensor reformulation; maintenance; topological changes to the hypercube might be considered or using the hypercube approach on a different topology is also possible; and multiple hypercube architectures.

5.1.4 Conclusions.

We stated in section 3.1 that one *raison d'etre* of this project was to assess the applicability of a truly scientific approach to non-symbolic learning as a necessary step towards limiting the number of "hidden variables" involved; and we feel our success in this endeavour to be encouraging. The completed scientific framework is:

1) The psychological experiments lead to our assumed fundamental fact that in the absence of disturbing factors the brain prefers the simplest possible induction consistent with the evidence.
2) The simplest possible theoretical machine which works in this way leads deductively, in the general case, to:
the theoretical subhypercube machine predictions.

3) A program was written to work according to the subhypercube theory and the theoretical predictions tested on it.
4) Pivotal aspects of this theory or its inverse were then assumed *incorrect* as a basis for experiments. These experiments after analysis led to:
the experimental result,
that it is incorrect to assume that the subhypercube theory is incorrect. Further experiments considered the scope, limitations and application of the practical implementation of the theoretical machine.

But this does *not* mean that the subhypercube theory is correct or even useful; merely that the theory should be retained pending further experiments, rather than discarded, as in the case of it being disproved. Hence the final seemingly weak conclusion of the project is that this work is worthy of further research. This is the correct scientific stance in the circumstances.

It is worth while saying that: the particular combination of advantages of our theory would have been very difficult to discover by any other means. Many aspects of this thesis are extremely interesting, even instructive. Hence the real power of this research lies in its pointers towards novel directions for future work.

REFERENCES.

Ahn, W., Mooney, R.J., Brewer, W.F. & Dejong, G.F., (1987), "Schema acquisition from one example: Psychological evidence for explanation-based learning", Tech. Rep. UILU-ENG-87-2231, Coordinated Science Laboratory, University of Illinois, Urbana-Champaign, Ill.

Aikins, J., (1983), "Prototypical knowledge for expert systems", *Artificial Intelligence*, **20**, pp.163-210.

Amarel, S., (1986), "Program Synthesis as a Theory Formation Task", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning. An Artificial Intelligence Approach, Volume 11,* Los Altos, CA: Morgan Kaufmann, pp. 499-569.

Amit, D.J., (1989), *Modeling Brain Function. The world of attractor neural networks,* New York: Cambridge University Press.

Anderson, J.A., (1977), "Neural Models with Cognitive Implications", in Laberage, D., (Ed.), *Basic processes in reading: perception and comprehension,* New York: Wiley, pp. 27-90.

Anderson, J.R., (1983), *The Architecture of Cognition,* Cambridge, Mass.: Harvard University Press.

Anderson, J.R., (1986), "Knowledge Compilation: The General Learning Mechanism", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning. An Artificial Intelligence Approach, Volume 11,* Los Altos, CA: Morgan Kaufmann, pp. 289-310.

Anderson, J.R. & Kline, P.J., (1979), "A learning system and its psychological implications", *Proceedings of the 6th International Joint Conference on Artificial Intelligence,* pp. 16-21.

Anderson, J.A. & Mozer, M.C., (1989), "Categorization and Selective Neurons", in Hinton, G.E. & Anderson, J.A., (Eds.), *Parallel Models of Associative Memory*, Updated Edition, Hillsdale, New Jersey: Lawrence Erlbaum Associates, pp.251-247.

Anderson, J.A. & Rosenfeld, E., (Eds.), (1989), *Neurocomputing Foundations of Research*, Cambridge, Mass.: MIT Press.

Anderson, J.A., Silverstein, J.W., Ritz, S.A. & Jones, R.S., (Eds.), (1989), "Introduction (1977)", in Anderson, J.A. & Rosenfeld, E., (Eds.), *Neurocomputing Foundations of Research*, Cambridge, Mass.: MIT Press.

Ball, D. & Harget, A.J., (1989), "Induction by a Hilbert Hypercube Representation," *Proceedings of the IASTED International Symposium Applied Informatics,* Anaheim: Acta Press, pp 1-4.

Ballard, D.H., (1986), "Cortical Connections and Parallel processing: Structure and Function", *The Behavioral and Brain Sciences*, **9**, pp. 67-120.

Batchelor, B.G., (1974), *Practical Approaches to pattern classification*, London: Plenum Books.

Belkin, N.J., Brooks, H.M. & Daniels, P.J., (1988), "Knowledge elicitation using discourse analysis", in Gaines, B.R. & Boose, J.H., *Knowledge Acquisition for Knowledge-Based Systems*, **1**, London: Academic Press, pp. 107-124.

Biederman, I., Hilton, H.J. & Hummel, J.E., (1991), "Pattern Goodness and Pattern Recognition" in Pommerantz, J.R. & Lockhead, G.R., (Eds.), *The Perception of Structure,* Washington, D.C: APA.

Birkhoff, G., (1948), *Lattice Theory,* New York: American Mathematical Society.

Blackemore, C., (1990), *The Mind Machine,* London: BBC Classics.

Boden, M.A., (1977), *Artificial Intelligence and Natural Man,* New York: Basic Books.

Bramer, M.A., (1987), "Automatic Induction of Rules from Examples: A Critical Analysis of the ID3 Family of Rule Induction Systems", Proceedings of the 1st European Workshop on Learning, F2.

Bratko, I., (1990), *PROLOG programming for Artificial Intelligence Second Edition,* International Computer Science Series, Wokingham, England: Addison-Wesley.

Bratko, I. Kononenko, I, Lavrac, N., Mozetic, I. & Roskar, E., (1985), "Automatic synthesis of knowledge", *Automatica* (Yugoslavia), **26**(3) pp171-175.

Braverman, M.S. & Russell, S.J., (1988), "Boundaries of operationality", *Proceedings of the 5th International Conference on Machine Learning,* Los Altos, Calif.: Morgan Kaufmann, pp. 221-234.

Brown, G.S., (1969), *Laws of Form*, London: Allen and Unwin.

Buchanan, B.G. & Feigenbaum, E.A., (1978), "DENDRAL and Meta-DENDRAL: their applications dimension", *Artificial Intelligence*, **11**, pp. 5-24.

Bunday, A. & Silver, B., (1982), "A critical survey of rule learning programs", *Proceedings of the European Conference on Artificial Intelligence,* pp. 150-157.

Caianiello, E.R., (1988), "Neuronic Equations and their Solutions", in Lee, Y.C., (Ed.), *Evolution, Learning and Cognition,* Singapore: World Scientific.

Campbell, J.A., (Ed.), (1985), *Implementations of Prolog,* Ellis Horwood Series Artificial Intelligence, New York: Ellis Horwood.

Carbonell, J.G., (1983), "Learning by Analogy: Formulating and Generalizing Plans from Past Experience", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach,* Palo Alto: Tioga, pp. 137-161.

Carbonell, J.G., (1986a), "Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning. An Artificial Intelligence Approach, Volume 11,* Los Altos, CA: Morgan Kaufmann, pp. 371-392.

Carbonell, J.G., (1986b), private communication with Carbonell.

Carbonell, J.G. & Langley, P (1986), "Learning and Knowledge Acquisition", in *Proceedings of the European Conference on Artificial Intelligence.*

Carbonell, J.G., Michalski, R.S., & Mitchell, T.M., (1983), "An Overview of Machine Learning", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach,* Palo Alto: Tioga, pp. 3-23.

Caruana, R.A. & Schaffer, J.D., (1988), Representation and Hidden Bias: Grey vs. Binary Coding for Genetic Algorithms", *Proceedings of the 5th International Conference on Machine Learning,* pp. 153-161.

Chen, M.S. & Shin, K.G., (1990), "Depth-First Search Approach for Fault Tolerant Routing in Hypercube Multicomputers", *IEEE Trans. on Parallel and Distributed Systems* **1**(2), pp.152-159.

Cheng, J., Fayyad, U.M., Irani, B.I. & Qian, Z., (1988), "Improved Decision Trees: A Generalised Version of ID3", *Proceedings of the 5th International Conference on Machine Learning*.

Cleaves, D.A., (1988), "Cognitive biases and corrective techniques: proposals for improving elicitation procedures for knowledge-based systems", in Gaines, B.R. & Boose, J.H., *Knowledge Acquisition for Knowledge-Based Systems*, **1**. London: Academic Press, pp. 23-34.

Cohn, P.M., (1965), *Universal Algebra*, New York: Harper and Row.

Conlon, T., (1989), *Programming in Parlog,* International Series in Logic Programming, Wokingham, England: Addison-Wesley.

Cordingley, E.S., (1989), "Knowledge elicitation techniques for knowledge-based systems", in Diaper, D., (ed), *Knowledge Elicitation: principles, techniques and applications*, Chichester, England: Ellis Horwood, pp.87-175.

Davis, R., (1980), "TEIRESIAS: applications of meta-level knowledge", in *Knowledge Based Expert Systems in Artificial Intelligence*, R. Davis & D. Lenat, New York: McGraw-Hill.

Davies, T.R. & Russell, S.J., (1987), "A logical approach to reasoning by analogy", *Proceedings of the 10th International Joint Conference on Artificial Intelligence,* Los Altos, Calif.: Morgan Kaufmann, pp. 264-270.

De Kleer, J., (1986), "Reasoning about multiple faults", *Proceedings of the 5th National Conference on Artificial Intelligence,* Los Altos, Calif.: Morgan Kaufmann, pp. 132-139.

Dejong, G.F., (1981), "Generalizations based on explanations", *Proceedings of the 7th International Joint Conference on Artificial Intelligence,* Los Altos, Calif.: Morgan Kaufmann, pp. 67-69.

Dejong, G.F. & Mooney, R., (1986), "Explanation-based learning: An alternative view", *Mach. Learn.*, **1**(2), pp. 145-176.

Delgrande, J.P., (1988), "A formal approach to learning from examples", in Gaines, B.R. & Boose, J.H., *Knowledge Acquisition for Knowledge-Based Systems*, **1**, London: Academic Press, pp. 163-181.

Diaper, D., (ed), (1989) *Knowledge Elicitation: principles, techniques and applications*, Chichester, England: Ellis Horwood.

Dietterich, T.G., (1986), "Learning at the knowledge level", *Machine Learning*, **1**(3), pp. 287-315.

Dietterich, T.G. & Michalski, R.S., (1981), "Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods", *Artificial Intelligence,* **16**, pp. 257-294.

Dietterich, T.G. & Michalski, R.S., (1983), "A Comparative Review of Selected Methods for Learning from Examples" in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach,* Palo Alto: Tioga, pp. 41-81.

Dijkstra, E.W., (1976), *A Discipline of programming*, Englewood Cliffs, N.J.: Prentice-Hall.

Duda, R.O., Gaschnig, J.G. & Hart, P.E., (1979), "Model design in the PROSPECTOR consultant system for mineral exploration", in Michie, D., (ed), *Expert systems in the micro-electronic age*, Edinburgh University Press, pp. 153-167.

Ehrlich, R., Crabtree, S.J., Kennedy, SK. & Cannon, R.L., (1984), "Petrographic image analysis of reservoir pore complexes", *Journal of Sedimentray Petrology*, 54(4) pp. 1365-1378.

Einstein, A., (1954), *Ideas and Opinions,* A Condor Book, London: Souvenir Press.

Elgalal , E.M.A., (1985), "Minimally-redundant data structures and reasonable hypotheses: some general heuristic methods of knowledge processing", Ph.D. Thesis, University of Strathclyde, Glasgow.

Ellman, T., (1989), "Explanation-Based learning: A Survey of Programs and Perspectives", *ACM Computing Surveys*, **21**(2).

Fahlman, S., (1988), private communication with the author.

Fahlman, S., (1989), "Parity benchmark", @edu.cmu.cs.gp.b, 16 Apr. 15:01:21 EDT.

Feigenbaum, E.A., McCorduck, P. & Nii, H.P., (1988), *The rise of the expert company,* Times Books.

Fikes, R.E., Hart, P.E. & Nilsson, N.J., (1972), "Learning and executing generalized robot plans", *Artificial Intelligence*, **3**, pp.251-288.

Fisher, D. & Langley, P., (1985), "Approaches to Conceptual Clustering", *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pp. 691-697.

Frisby, J.P., (1979), *SEEING Illusion, Brain and Mind*, Oxford: Oxford University Press.

Fukushima, K., Miyake, S. & Ito, K., (1983), "Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition", IEEE Transactions on Systems, Man and Cybernetics, **SMC-13**(5).

Gaines, B.R., (1988), "An overview of knowledge-acquisition and transfer", in Gaines, B.R. & Boose, J.H., *Knowledge Acquisition for Knowledge-Based Systems*, **1**, London: Academic Press, pp. 3-22.

Gaines, B.R. & Boose, J.H., (1988), "Knowledge Acquisition for Knowledge-Based Systems", in Gaines, B.R. & Boose, J.H., *Knowledge Acquisition for Knowledge-Based Systems*, **1**, London: Academic Press, pp. xiii-xix.

Gallant, S.I., (1990), "A Connectionist Learning Algorithm with Provable Generalisation and Scaling Bounds", Neural Networks, **3**, pp. 191-201.

Ganascia, J.G., (1987a) "Charade: A Rule System Learning System" *Proceedings of the 10th International Joint Conference on Artificial Intelligence,* Los Altos, Calif.: Morgan Kaufmann, pp. 345-347.

Ganascia, J.G., (1987b) "Learning with Hilbert Cubes", *Proceedings of the 1st European Workshop on Learning,* pp. 158-171.

Ganascia, J.G., (1988), "Improvement and Refinement of the Learning Bias Semantic", *Proceedings of the 8th European Conference on Artificial Intelligence,* Munich, pp.384-389.

Gangly, B., (1987), "The Devolving Science of Machine Learning", *Proceedings of the 4th International Workshop on Machine Learning,* pp. 398-401.

Giarratano, J., (1989), *Expert systems principles and programming*, Boston: Pws-kent.

Golden, R.M., (1986), "The "brain-state-in-a-box" neural model is a descent algorithm", *Journal of Mathematical Psychology,* **30**(1), pp.73-80.

Gupta, A (1988), "Significance of the explanation language in EBL", *Proceedings of the AAAI Spring Symposium on Explanation-Based Learning, Menlo Park, Calif.: AAAI,* pp. 73-77.

Hass, N. & Hendrix, G.G., (1983), "Learning by being told: acquiring knowledge for information management", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach,* Palo Alto: Tioga, pp. 405-427.

Haussler, D., (1988), "Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework", *Artificial Intelligence,* **36**, pp.177-221.

Hayes-Roth, F., Waterman, D.A. & Lenat, D.B., (Eds.), (1983), *Building Expert Systems*, London: Addison-Wesley.

Hebb, D.O., (1949), *The Organization of Behaviour,* John Wiley.

Hillis, W.D., (1987), *The Connection Machine*, Cambridge Mass.: MIT Press.

Hinton, G.E., Sejnowski, T.J. & Ackley, D.H., (1984), "Boltzmann Machines: Constraint Satisfaction Networks that Learn", Technical report CMU-CS-84-119, Computer Science Department, Carnegie-Mellon University.

Hinton, G.E. & Anderson, J.A., (Eds.), (1989), *Parallel Models of Associative Memory*, Updated Edition, Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Holland, J., (1975), *Adaptation in Natural and Artificial Systems,* Ann Arbor, Michigan: University of Michigan Press.

Holland, J.H., (1986), "Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning. An Artificial Intelligence Approach, Volume 11,* Los Altos, CA: Morgan Kaufmann, pp. 593-623.

Holland, J.H., (1988), "The Dynamics of Searches Directed by Genetic Algorithms", in Lee, Y.C., (Ed.), *Evolution, Learning and Cognition,* Singapore: World Scientific.

Hubel, D.H., (1988), *Eye, Brain and Vision,* Scientific American Library, New York: Freeman.

Hubel, D.H., Wiesel, T.N. & Stryker, M.P., (1978), "Anatomical demonstration of orientation columns in macaque monkey", *Journal of Comparative Neurology,* **177**(3), pp. 361-380.

Hubel, D.H. & Wiesel, T.N., (1979), "Brain Mechanisms of Vision," in Scientific American, September, pp. 146-156.

Huyser, K.A. & Horowitz, M.A., (1988), "Generalization in Connectionist Networks that Realise Boolean Functions", *Proceedings of the 1988 Connectionist Models Summer School,* pp. 191-200.

Iversen, L.L., (1979), "Brain Mechanisms of Vision," in Scientific American, September, pp. 118-129.

Jackson, P., (1986), *Introduction to expert systems*, International Computer Science Series, Wokingham, England: Addison-Wesley.

Kahn, G.S., (1986), "Knowledge Acquisition: Investigations and General Principles", in Mitchell, T.M., Carbonell, J.G., & Michalski, R.S., (Eds.), *Machine Learning: A Guide to Current Research,* Hingham, MA.: Kluwer Academic Press, pp. 119-121.

Kandel, A. & Lee, S.C., (1979), "Fuzzy Switching and Automata", Theory and Applications: Theory and Applications, London: Edward Arnold.

Karmarker, N., (1984a), "A New Polynomial-Time Algorithm for Linear Programming", *Proceedings of the 16th Annual ACM Symposium on the Theory of Computing,* Washington D.C., pp. 302-311.

Karmarker, N., (1984b), "A New Polynomial-Time Algorithm for Linear Programming", *Combinatorica* **4**(4), pp. 373-395.

Kelly, G.A., (1955), *The Psychology of Personal Constructs*, New York: Norton.

Keller, R.M., (1987), "The role of explicit contextual knowledge in learning concepts to improve performance", Ph.D. thesis and Tech. Rep. ML-TR-7, Department of Computer Science, Rutgers University, New Brunswick, N.J.

Kidd, A.L., (1986), "Knowledge elicitation", *Proc. of the 6th Technical Conference of the BCS specialist group on expert systems,* Expert Systems 86, Brighton.

Kodratoff, Y. & Ganascia, J., (1986), "Improving the generalisation step in learning", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning. An Artificial Intelligence Approach, Volume 11,* Los Altos, CA: Morgan Kaufmann, pp. 215-244.

Kodratoff, Y., (1988), *Introduction to Machine Learning*, London: Pitman.

Kuipers, B. & Kassiver, J.P., (1983), "How to discover a knowledge representation for causal reasoning by studying an expert physician", *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Karlsruhe, pp. 49-56.

LaFrance, M., (1988), "The Knowledge Acquisition Grid: a method for training knowledge engineers", in Gaines, B.R. & Boose, J.H., *Knowledge Acquisition for Knowledge-Based Systems*, **1**, London: Academic Press, pp. 81-91.

Laird, J.E., Rosenbloom, P.S. & Newell, A., (1986), "Chunking in SOAR: The Anatomy of a General Learning Mechanism", *Machine Learning*, **1**.

Langley, P.W., Simon, H.A & Bradshaw, G.L., (1983), "Rediscovering Chemistry with the BACON System", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach,* Palo Alto: Tioga, pp. 307-329.

Langley, P., Zytkow, J.M., Simon, H.A. & Bradshaw, G.L., (1986), "The Search for Regularity: Four Aspects of Scientific Discovery", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning. An Artificial Intelligence Approach, Volume 11,* Los Altos, CA: Morgan Kaufmann, pp. 425-469.

Lebowitz, M., (1986), "Concept Learning in a Rich Input Domain: Generalisation-Based Learning", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning.*

An Artificial Intelligence Approach, Volume 11, Los Altos, CA: Morgan Kaufmann, pp. 193-214.

Lenat, D.B., (1977), "Automated Theory Formation in Mathematics", *Proceedings of the 5th International Joint Conference on Artificial Intelligence,* pp. 833-842.

Lenat, D.B., (1983), "The Role of Heuristics in Learning by Discovery: Three Case Studies" in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach,* Palo Alto: Tioga, pp. 243-306.

Lindsay, R.K., Buchanan, B.G., Feigenbaum, E.A. & Lederberg, J., (1980), *Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project,* New York: McGraw-Hill.

Linsker, R., (1988), "Self Organization in a Perceptual Network", *Computer,* March 1988.

Lipschutz, S., (1964), *Schaum's Outline of Theory and Problems of Set Theory and Related Topics,* Schaum's Outline Series, New York: McGraw-Hill Book Company.

Lowe, D. G., (1985), *Perceptual Organisation and Visual Recognition,* Kluwer Academic Publishers.

Lowe, D. G., (1987), "Three Dimensional Object Recognition from Single Two Dimensional Images.", *Artificial Intelligence.*, volume *31*, pp. 355-395.

Lowton, A., (1992), forthcoming Ph.D. thesis, Department of Computer Science, The University of Aston in Birmingham, U.K.

Manago, M.V. & Kodratoff, Y., (1987), "Noise and Knowledge Acquisition", *Proceedings of the 10th International Joint Conference on Artificial Intelligence,* Los Altos, Calif.: Morgan Kaufmann.

Markovitch, S & Scott, P.D., (1988), "The role of forgetting in learning", *Proceedings of the 5th International Conference on Machine Learning,* Los Altos, Calif.: Morgan Kaufmann, pp. 459-465.

McClelland, J.L. & Rumelhart, D., (1988), *Explorations in Parallel Distributed Processing A Handbook of Models, Programs and Exercises,* Cambridge, MA: MIT Press.

McCorduck, P., (1979), *Machines who think,* San Francisco: Freeman.

McDermott, J., (1981), "R1: The formative years", *AI Magazine* **2**(2), pp. 21-29.

Michalski, R.S., (1980), "Pattern recognition as rule-guided inductive inference", *IEEE Trans. Pattern Anal. Mach. Intell.* **2**(4), pp.349-361.

Michalski, R.S., (1983), "A Theory and Methodology of Inductive Learning" in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach,* Palo Alto: Tioga, pp. 83-134.

Michalski, R.S. & Chilausky, R.L., (1980), "Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis", Policy Analysis and Information Systems, *4*(2).

Michalski, R.S. & Stepp, R.E., (1983), "Learning from Observation: Conceptual Clustering", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach,* Palo Alto: Tioga, pp.331-363.

Michalski, R.S., Carbonell, J.G. & Mitchell, T.M., (Eds.), (1983), *MACHINE LEARNING An Artificial Intelligence Approach*, Palo Alto: Tioga.

Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), (1986), *Machine Learning. An Artificial Intelligence Approach, Volume 11*, Los Altos, CA: Morgan Kaufmann.

Minsky, M., (1986), *Society of Mind*, New York: Simon and Schuster.

Minsky M.L. & Papert S.A., (1969), *Perceptrons: An Introduction to Computational Geometry*, Cambridge, Massachusetts: The MIT Press.

Minsky M.L. & Papert S.A., (1988), *Perceptrons: An Introduction to Computational Geometry, Expanded Version*, Cambridge, Massachusetts: The MIT Press.

Minton, S., (1985), "Selectively generalizing plans for problem-solving", *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Los Altos, Calif.: Morgan Kaufmann, pp. 596-599.

Minton, S., (1988), "Quantitative results concerning the utility of explanation-based learning", *Proceedings of the 7th National Conference on Artificial Intelligence*, Los Altos, Calif.: Morgan Kaufmann, pp. 564-569.

Misner, C.W., Thorne, K.S. & Wheeler, J.A., (1973), *Gravitation*, San Francisco: Freeman.

Mitchell, T.M., (1978), "Version spaces: An approach to concept learning", Tech. Rep. HPP-79-2 Computer Science Department, Stanford University, Palo Alto, Calif.

Mitchell, T.M., (1980), "The need for biases in learning generalizations", Tech. Rep. CBM-TR-117, Department of Computer Science, Rutgers University, New Brunswick, N.J.

Mitchell, T.M., (1982), "Generalization as search", *Artificial Intelligence*, **18**, pp. 203-226.

Mitchell, T.M., (1983), "Learning and problem solving", *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Los Altos, Calif.: Morgan Kaufmann pp. 1139-1151.

Mitchell, T.M., (1985), "LEAP: A learning apprentice system for VLSI design", *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Los Altos, Calif.: Morgan Kaufmann pp. 573-580.

Mitchell, T.M., Utgoff, P.E. & Banerji, R., (1983), "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics" in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach*, Palo Alto: Tioga, pp. 163-190.

Mitchell, T.M., Keller, R.M. & Kedar-Cabelli, S.T., (1986), "Explanation-Based Generalization: A Unifying View", *Machine Learning*, **1**, pp. 47-80.

Mooney, R.J., (1988), "Generalizing the Order of Operators in Macro-Operators", *Proceedings of the 5th National Conference on Machine Learning*, pp270-283.

Mooney, R.J., Shavlik, J., Towell, G. & Gove, A. (1989), "An Experimental Comparison of Symbolic and Connectionist Learning Algorithms", *Proceedings of the 11th International Conference on Artificial Intelligence*, pp. 775-780.

Mostow, D.J., (1983), "Machine Transformation of Advice into a Heuristic Search Procedure", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach*, Palo Alto: Tioga, pp. 367-403.

Mostow, J., (1983), "Operationalizing advice: A Problem-solving model", *Proceedings of the International Workshop on Machine Learning,* Department of Computer science, University of Illinois, Urbana, Ill., pp. 110-116.

Neumann, J.V., (1958), *The Computer and the Brain,* New Haven: Yale University Press.

Newell, A., (1980), "Reasoning, problem solving, and decision processes: The problem space as a fundamental category", in Nickerson, (ed), *Attention and Performance*, Hilladale, N.J.: Erlbaum.

Newell, A. & Simon, H.A., (1963), "GPS, A Program that Simulates Human Thought", in Feigenbaum, E.A. & Feldman, J., (Eds.), *Computers and Thought,* New York: McGraw-Hill.

Nilsson, N.J., (1980), *Principles of Artificial Intelligence*, Palo Alto, Calif.: Tioga.

Oosthuizen, G.D., (1986), "A Paradigm for Automatic Learning", Internal Report FACT 24/86, University of Strathclyde, Scotland.

Oosthuizen, G.D., (1987a) "Graph Induction", *Proceedings of the 1st European Workshop on Learning,* pp. 158-171.

Oosthuizen, G.D., (1987b), "SUPERGRAN: a connectionist approach to learning integrating genetic algorithms and graph induction", *Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications*, MIT, Cambridge, MA, pp. 132-139.

Oosthuizen, G.D. & McGregor, D.R., (1988), "Induction through knowledge Base Normalisation", *Proceedings of the 8th European Conference on Artificial Intelligence,* Munich, pp. 396-401

Pople, H.E., (1977), "The formation of composite hypotheses in diagnostic problem solving: An exercise in synthetic reasoning", *Proceedings of the 5th International Joint Conference on Artificial Intelligence,* pp. 1030-1037.

Popper, K.R., (1974), *Conjectures and Refutations The Growth of Scientific Knowledge*, London: Routledge and Kegan Paul.

Quinlan, J.R., (1983), "Learning Efficient Classification Procedures and their Application to Chess End Games" in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *MACHINE LEARNING An Artificial Intelligence Approach,* Palo Alto: Tioga, pp. 463-482.

Quinlan, J.R., (1986), "The effect of noise on concept learning", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning. An Artificial Intelligence Approach, Volume 11,* Los Altos, CA: Morgan Kaufmann, pp. 149-166.

Quinlan, J.R., (1988), "An Empirical Comparison of Genetic and Decision-Tree Classifiers", *Proceedings of the 5th International Conference on Machine Learning,* pp. 135-141.

Ratcliff, B., (1987), *Software Engineering: Principles and Methods,* Computer Science Texts, Oxford: Blackwell Scientific Publications.

Rennels, G.D. & Shortliffe, E.H., (1987), "Advanced computing for medicine" in Scientific American, October, pp. 146-153.

Rose, D. & Dobson, V.G., (Eds.), (1985), *Models of the Visual Cortex,* Wiley-Interscience, Chichester: Wiley.

Rosenblatt, F., (1962), *Principles of Neurodynamics,* Spartan Books.

Rosenbloom, P.S. & Newell, A., (1986), "The Chunking of Goal Hierarchies: A Generalized Model of Practice", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning. An Artificial Intelligence Approach, Volume 11,* Los Altos, CA: Morgan Kaufmann, pp. 247-288.

Roth, I. & Frisby, J.P., (1986), *Perception and Representation a Cognitive Approach,* Open Guides to Psychology, Milton Keynes: Open University.

Rucker, R.V.B., (1977), *Geometry, Relativity and the Fourth Dimension,* New York: Dover Publications.

Russell, S.J., (1986), "Preliminary steps toward the automation of induction", *Proceedings of the 5th National Conference on Artificial Intelligence,* Los Altos, Calif.: Morgan Kaufmann, pp. 477-484.

Russell, S.J. & Grosof, B.N., (1987), "A declarative approach to bias in concept learning", *Proceedings of the 6th National Conference on Artificial Intelligence,* Los Altos, Calif.: Morgan Kaufmann, pp. 505-510.

Rumelhart, D. & McClelland, J.L., (Eds.), (1986a), *Parallel Distributed Processing Explorations in the Microstructure of Cognition Vol 1: Foundations,* Cambridge, MA: MIT Press.

Rumelhart, D. & McClelland, J.L., (Eds.), (1986b), *Parallel Distributed Processing Explorations in the Microstructure of Cognition Vol 2: Psychological and Biological Models,* Cambridge, MA: MIT Press.

Sarkaria, S.S., (1990), "Neural Networks for Perceptual Grouping," Ph.D. thesis, Department of Computer Science, The University of Aston in Birmingham, U.K.

Schlimmer, J.C. & Fisher, D., (1986), "A case study of incremental concept induction", *Proceedings of the 5th National Conference on Artificial Intelligence,* Morgan Kaufmann, pp.496-501.

Selfridge, O.G., (1959), "Pandemonium: a Paradigm for Learning," in Blake, D. & Uttley, A., (Eds.), *Proc. Symposium on Mechanization of Thought Processes,* London: H.M. Stationary Office.

Shapiro, A.D., (1987), *Structured induction in expert systems,* Maidenhead: Addison Wesley.

Shortliffe, E.H., (1976), *Computer-based medical consultation: MYCIN,* New York: American Elsevier.

Steels, L., (1986), "Second Generation Expert Systems", *Proc. of the 6th Technical Conference of the BCS specialist group on expert systems,* Expert Systems 86, Brighton, pp. 175-183.

Stepp, R.E. & Michalski, R.S., (1986), "Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning. An Artificial Intelligence Approach, Volume 11,* Los Altos, CA: Morgan Kaufmann, pp. 471-498.

Stonier, T., (1986) "What is knowledge?", *Proc. of the 6th Technical Conference of the BCS specialist group on expert systems,* Expert Systems 86, Brighton, pp. 217-230.

Tambe, M. & Newell, A., (1988), "Some chunks are expensive" *Proceedings of the 5th International Conference on Machine Learning,* Los Altos, Calif.: Morgan Kaufmann, pp. 451-458.

Tesauro, G., & Janssens, B., (1988), "Scaling Relationships in Back-Propagation Learning", *Complex systems 2*, pp.39-84.

Touretzsky, D.S., (1986), *The Mathematics of Inheritance Systems,* Research Notes in Artificial Intelligence, London: Pitman.

Turner, M., (1985), *Expert systems: A Management Guide*, PA Computers and Telecommunications.

Utgoff, P.E., (1986), "Shift of bias for inductive concept learning", in Michalski, R.S., Carbonell J.G. & Mitchell, T.M., (Eds.), *Machine Learning. An Artificial Intelligence Approach, Volume 11,* Los Altos, CA: Morgan Kaufmann, pp. 107-148.

Utgoff, P.E., (1988), "ID5: An Incremental ID3", *Proceedings of the 5th International Conference on Machine Learning.*

Utgoff, P.E. & Mitchell, T.M., (1982), "Acquisition of appropriate bias for inductive concept learning", *Proceedings of the National Conference on Artificial Intelligence,* Los Altos, Calif.: Morgan Kaufmann, pp. 414-417.

Valiant, L.G., (1985), "Learning Disjunctions of Conjunctions", *Proceedings of the 5th International Joint Conference on Artificial Intelligence.*

Vere, S.A., (1980), "Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions", *Artificial Intelligence*, **14,** pp.139-164.

Wasserman, P.D., (1989), *Neural Computing Theory and Practice*, New York: Van Nostrand Rienhold.

Welbank, M., (1983), "A review of knowledge acquisition techniques for expert systems", *British Telecom. Research Laboratories Report,* Martlesham Heath: Martlesham Consultancy Services.

Weiss, S.M., Kulikowski, C.A & Safir, A., (1978), "A model-based method for computer-aided medical decision-making", *Artificial Intelligence*, **11**, pp. 145-172.

Whitfield, I.C., (1984), *Neurocommunications: An Introduction,* A Whiley-Interscience Publication, Chichester: John Wiley.

Wilson, M., (1989), "Task models for knowledge elicitation", in Diaper, D., (ed), *Knowledge Elicitation: principles, techniques and applications*, Chichester, England: Ellis Horwood.

Wilson, S.W., (1987), "Quasi-Darwinian Learning in a Classifier system", *Proceedings of the 4th International Workshop on Machine Learning,* pp. 59-65.

Winograd, T. & Flores, F., (1986), *Understanding Computers and Cognition: A New Foundation for Design,* New Jersey: Ablex Publishing Corporation.

Winston, P.H., (1975), "Learning Structural Descriptions from Examples", in Winston, P., (ed), *The Psychology of Computer Vision,* New York: McGraw-Hill.

Winston, P.H., (1985), *Artificial Intelligence*, second edition, Reading, MA.: Addison-Wesley.

Wirth, J. & Catlett, J., (1988), "Experiments on the Costs and Benefits of Windowing in ID3", *Proceedings of the 5th International Conference on Machine Learning.*

Wisniewski, E.J. & Anderson, J.A., (1988), "Some Interesting Properties of a Connectionist Inductive Learning System", *Proceedings of the 5th International Conference on Machine Learning,* Los Altos, Calif.: Morgan Kaufmann, pp. 181-187.

Youssef, A.S. & Narahari, B., (1990), "The Banyan-Hypercube Networks", *IEEE Trans. on Parallel and Distributed Systems* **1**(2), pp.160-169.

Zytkow, J.M. & Simon, H.A., (1986), "A Theory of Historical Discovery: The Construction of Componential Models", *Machine Learning*, **1**.

APPENDIX 1

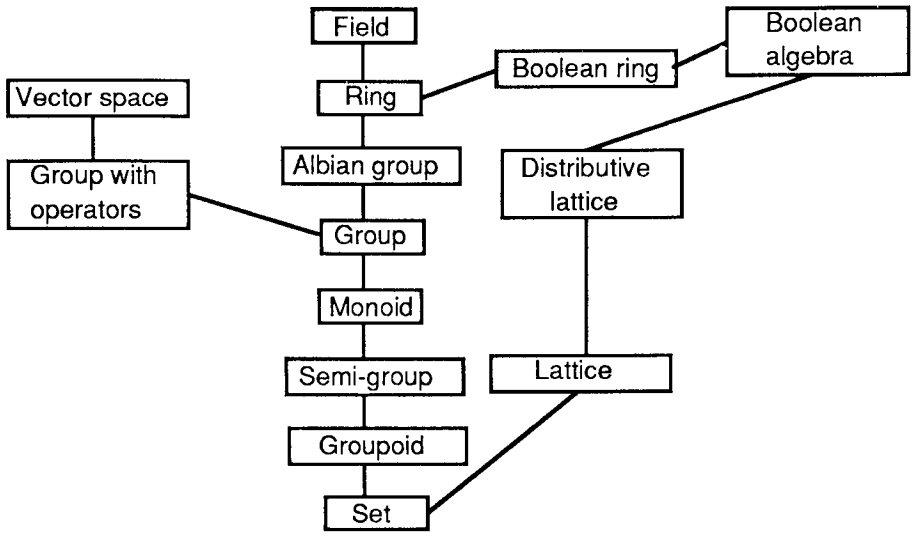
Notation.

A, B, ...	sets
A'	complement of set A
$\{\alpha, \beta, ...\}$	set of elements α, β, ...
{a, e, i, o, u}	defined by listing all members, tabular form of set
{x ∋ x is vowel}	defined by property of all members, set of all x such that x is a vowel
$\alpha \in A$	object α is an element of set A, α belongs to A
$\alpha \notin A$	object α is not an element of set A, α does not belong to A
α^{-1}	inverse element of α
$\varnothing$	null set, set with no elements
Θ	universal set
$A \supseteq B$	A contains or includes B, B is a subset of A
$A \cap B$	intersection
$A \cup B$	union
$\alpha \Re \beta$	α is related to β
$\alpha \Re' \beta$	α is not related to β
$A \otimes B$	2-tuple tensor of rank two as cartesian outer/cross product
$\Rightarrow$	implication
$\wedge$, $\vee$, ϑ	logical: and, or, exclusive or
$\exists$, $\forall$	logical: there exists, for all
ε	quotient set
$\wp$	is uniquely an element of, belongs to one and only one of
iff	if and only if
Ξ	partial order relation analogous to less than or equal to
$\Im$, $\aleph$	general binary operators
$(A, \Im)$	groupoid
$(A, \Im, \aleph)$	ring
$\langle \alpha, \beta \rangle$	ordered pair
sup A	supremum or least upper bound of set A
inf A	infimum or greatest lower bound of set A
R	set of all real numbers
Φ	Φ–boolean quantity
∋	such that, where

APPENDIX 2

Definitions

Algebraic systems: These are related as given diagramatically below.



Antisymmetric relation: A relation $\Re$ on set A is antisymmetric if: $\alpha \Re \beta \wedge \beta \Re \alpha \Rightarrow \alpha = \beta \ni \alpha, \beta \in A$.

Associative memory: An associative memory is based on only partial or partially erroneous information.

Associative binary operation: An associative binary operation is a closed binary operation $\Im$ on set A: $\ni (\alpha \Im \beta) \Im \gamma = \alpha \Im (\beta \Im \gamma) \forall \alpha, \beta, \gamma \in A$.

Bijection: A bijection is a one-to-one onto function.

Binary operation: A binary operation on set A assigns a unique element β to all ordered pairs of elements $\langle \alpha, \alpha \rangle \in A \otimes A$ and the set A is closed under this operation iff $\forall \beta \in A$.

Binary relation: A binary relation $\Re$ between sets A and B assigns: $\forall \langle \alpha, \beta \rangle \in A \otimes B \exists \wp (\alpha \Re \beta \vartheta \alpha \Re' \beta)$.

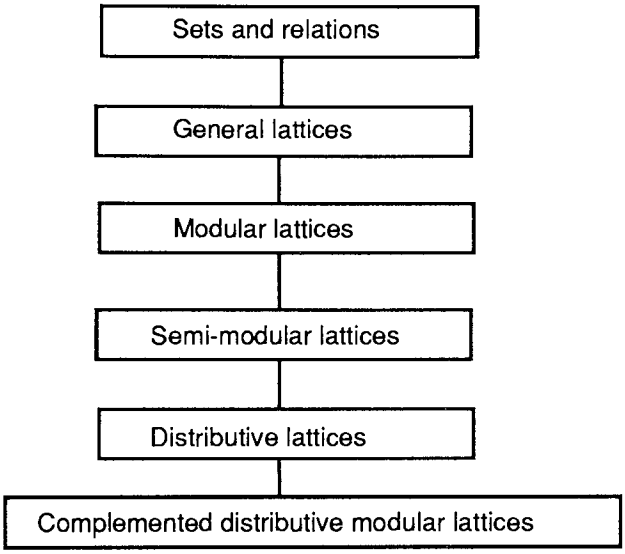
Bounds: Limitations of a set. Of particular interest in lattice theory are the "greatest lower bound" and the "least upper bound" of a set. See below.

Closed binary operation: A closed binary operation $\Im$, on set A $\Rightarrow$ set A is closed under the operation $\Im$, iff set A assigns a unique element $\beta \forall \beta \in A$ to all ordered pairs of elements $\langle \alpha, \alpha \rangle \in A \otimes A$.

Commutative binary operation: A commutative binary operation $\Im$, on set A has: $\alpha \Im \beta = \beta \Im \alpha \; \forall \; \alpha, \beta \in A$

Commutative group: A commutative group is a group which has a commutative binary operation.

Complemented distributed modular lattice or CDML or boolean lattice: The algebraic systems are related as above. Within this system restrictions upon the lattice lead to CDML as below.



Converse relation: The converse relation ® of a given relation $\Re$ between sets A and B is the set of all reversely ordered solution set ordered pairs: $\langle \beta, \alpha \rangle \in B \otimes A \ni \beta \circledR \alpha \; \forall \; \langle \alpha, \beta \rangle \in A \otimes B \ni \alpha \Re \beta$.

Correspondence: A correspondence from the codomain of set B relates each element of the domain A to one or more elements of B.

Defining special relations: Either by defining the set and relationship or by defining the set and the solution set of the relation.

Distributive binary operation: A distributive binary operation is a closed commutativebinary operation $\Im$ on set A which is said to be distributive over a closed binary operation $\aleph$ on A iff: $\alpha \Im (\beta \aleph \gamma) = (\alpha \Im \beta) \aleph (\alpha \Im \gamma) \; \forall \; \alpha, \beta, \gamma \in A$.

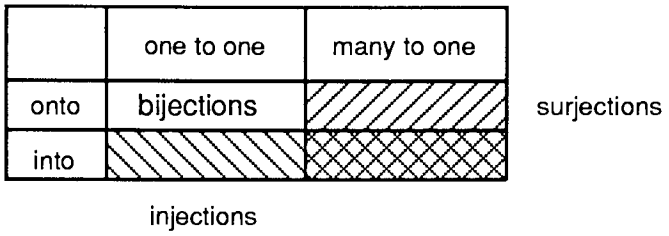
Equal sets: Two sets A and B are said to be equal if they consist of the same elements. That is: $A = B$ iff $\forall \; \alpha \; (\alpha \in A, B) \wedge \forall \; \beta \; (\beta \in A, B)$.

Equivalence class: The equivalence class of set A is a subset belonging to a partition of A by an equivalence relation.

Equivalence relation: A relation $\Re$ on set A which is reflexive, symmetric and transitive $\ni \Re$ partitions A into non-overlapping subsets of: $\varepsilon \ni \alpha \wp \varepsilon \forall \alpha \in A$.

Field: A field is a commutative ring (A, $\Im$, $\aleph$) with an identity and the further property that, excepting the identity for $\Im$, every element has an inverse with respect to $\aleph$.

Functions: A function is a correspondence which is either one-to-one or many-to-one. The types of function are: bijections, injections and surjections as in the diagram below.



Greatest lower bound: The greatest lower bound (glb) of subset B of a partially ordered set A, the infimum of B written as inf B, is that lower bound α where $\alpha \in B$ for every lower bound β of B: $\ni \beta \Xi \alpha \forall \beta \in B$.

Group: A group is a monoid with the additional property given identity e, an inverse α^{-1} and binary operation $\Im$ that: $\forall \alpha \in A \exists \alpha^{-1} \ni \alpha \Im \alpha^{-1} = \alpha^{-1} \Im \alpha = e$

Groupoid: A groupoid is a set with a closed binary operation defined on it.

Hasse diagram: A hasse diagram is a diagram depicting an order relation.

Ideal: An ideal is a subring (B, $\Im$, $\aleph$) of a ring (A, $\Im$, $\aleph$) with the property that $\alpha \Im \beta \in B$ and $\beta \Im \alpha \in B \forall \alpha \in A, \beta \in B$.

Identity element: The identity element e for the binary operation $\Im$ on set A is: $e \wp A \ni e \Im \alpha = \alpha \Im e \forall \alpha \in A$.

Image: The image of element α of the domain is the set of all elements of the codomain of a correspondence which are related to α.

Injection: An injection is a one-to-one function. See function.

Into: An into correspondence is a correspondence such that the image of the domain is a proper subset of the codomain.

Inverse function: The inverse function is the converse of a bijection.

Isomorphism: A morphism which is one to one and onto (see bijection and function).

Kernel of a morphism: The kernel of a morphism is that subset of the domain of a morphism such that the image is the identity of the codomain

Lattice: A lattice is a set with a partial order relation defined on it such that any subset of two elements has both a greatest lower bound and a least upper bound.

Least upper bound: The least upper bound (lub) of subset B of a partially ordered set A, the supremum of B written as sup B, is that upper bound α where $\alpha \in B$ for every upper bound β of B: $\ni \alpha \, \Xi \, \beta \, \forall \, \beta \in B$.

Lower bound: A lower bound of subset B of a partially ordered set A is any element: $\alpha \in A \ni \alpha \, \Xi \, \beta \, \forall \, \beta \in B$

Many-to-many: A many-to-many correspondence has a unique image for at least one element of the domain comprising of more than one element of the codomain and at least one element of the codomain is the image of more than one element of the domain.

Many-to-one: A many-to-one correspondence has a unique image for each element of the domain and at least one element of the codomain is the image of more than one element of the domain.

Learning metric: A learning machine whose polytopes have a mathematically definable relationship to the universe. The consequent advantages of a metric learning machine are reliability, dependability and reproducability; hence leading to **predictable behaviour**.

(We have in mind here the idea that such a learning machine allows us to know where it is searching in the space by reference to a metric related to the **universe**: namely a given subhypercube or in the general case a polytope. Here is the crucially important advantage over other connectionist and general learning methods as seen for example in section 3.2.5. In this regard we have a **novel third paradigm** as an approach to the speed and complexity problem. See also section 4.3.2).

Monoid: A monoid is a semigroup with an identity.

Morphism: A morphism is a function from (A, $\Im$) to (B, $\aleph$) such that the structure (B, $\aleph$) models the structure of (A, $\Im$) and hence $\Rightarrow$ that the image of $\alpha \, \Im \, \beta$ equals the combined images of α and β under $\aleph \, \forall \, \alpha, \beta \in A$.

Nodes: The elements in a space. (See space).

One-to-many: A one-to-many correspondence has a unique image for each element of the domain comprising of more than one element of the domain and each element of the codomain is the image of a unique element of the domain.

One-to-one: A one-to-one correspondence has a unique image for each element of the domain and no element of the codomain is the image of more than one element of the domain.

Onto: An onto correspondence is a correspondence such that each element of the codomain is an image of at least one element of the domain, the image of the domain being the complete codomain.

Ordered set: An ordered set is a set with an order relation defined on it.

Partial order relation: A partial order relation is a relation on a set which is reflexive, antisymmetric and transitive.

Partition of a set: A partition ρ of set A separates the elements α into subsets: $\ni \alpha \wp \rho \forall \alpha \in A$.

Polytope: Any grouping or groupings of subhypercubes defining class 1, true or positive.

Quotient set: The quotient set ε of a set with an equivalence relation defined on it is the set of all equivalence classes.

Reflexive relation: A relation $\Re$ on set A is reflexive if: $\alpha \Re \alpha \forall \alpha \in A$.

Ring: A ring $(A, \Im, \aleph)$ is a set A with two binary operations $\Im$, $\aleph$ defined on it such that $(A, \Im)$ is a commutative group, $(A, \aleph)$ is a semigroup and $\aleph$ is distributive over $\Im$.

Semigroup: A semigroup is a set with a closed associative binary operation defined on it.

Solution set: The Solution set of relation $\Re$ between sets A and B is the set of all ordered pairs: $\langle \alpha, \beta \rangle \in A \otimes B \ni \alpha \Re \beta$

Space: A space is any non-empty set which possesses some type of mathematical structure - as in vector space, metric space or topological space.

Structure: A structure is a set with one or more operations and/or relations defined on it.

Surjection: A surjection is an onto function.

Symmetric relation: A relation $\Re$ on set A is symmetric if: $\alpha \Re \beta \forall \beta \Re \alpha \ni \alpha, \beta \in A$.

The cartesian product: The cartesian product of sets A and B, namely $A \otimes B$, is the set of all ordered pairs of elements $\langle \alpha, \beta \rangle \ni \alpha \in A, \beta \ni B$.

The Power set: The power set of set A is the set of all subsets of A.

Total order relation: A total order relation between sets A and B is a partial order relation $\Re$ on A: $\ni \forall \langle \alpha, \beta \rangle \in A \otimes B \ni \alpha \neq \beta \exists \wp (\alpha \Re \beta \vartheta \beta \Re \alpha)$.

Transitive relation: A relation $\Re$ on set A is transitive if: $\alpha \Re \gamma \forall (\alpha \Re \beta \wedge \beta \Re \gamma) \ni \alpha, \beta, \gamma \in A$.

Upper bound: A upper bound of subset B of a partially ordered set A is any element: $\alpha \in A \ni \beta \Xi \alpha \forall \beta \in B$

APPENDIX 3.

Complexity Analysis.

What distinguishes P and NP is how the time needed for a solution grows, as the size of the problem increases. If the running time grows no faster than some fixed power of the size of the input data then the algorithm is said to run in polynomial time. These problems are called "easy" since large problems can be solved.

Thus: "easy" => dt approximates n^c => polynomial time (or P)

Conversely, problems that cannot be solved in polynomial time are called "hard". The running time increases exponentially in that increasing the size of the problem then multiplies the solution time equivalently. "Hard" problems can only be solved for small problems.

Thus: "hard" => dt approximates c^n => non-deterministic polynomial time (or NP)

e.g. For the hypercube the complete solution: 2^n => NP,
but note that for Resolution: $(a + x)^c$ => P.

Multidimensional analysis has been found to be very useful in the solution of hard problems. One of the best known examples is the work of Karlmaker for the problem of efficient allocation of resources (Karmarker, 1984a, 1984b). This is a very common problem in industry, hence Karmarker's work is widely used.

APPENDIX 4

Parallel architectures

It would obviously be advantageous if a parallel architecture implementation could speed up the processing of the HYPERCUBE thereby allowing much larger problems to be tackled. Hardware hypercube machines, such as the Intel NCUBE (not to be confused with the very different *learning* hypercube architecture described herein), have the advantage that they do not require switching networks or processors, between processors and memories. Each processor/memory occupies a corner of a hypercube lattice defined in hardware. Hardware hypercubes are very efficient for certain types of algorithm, where, for example, each processor in the lattice would be asked to model a small section of the system.

At first sight there seems to be a possibility that a hardware hypercube machine could provide the ideal architecture for implementing a parallel version of the search algorithm. This assumption, however, has not been investigated and it is even possible that a network of processors (e.g. transputers) which have excellent pattern recognition capabilities for implementing tree searches, or the "connectionist machine" with its capability for fine grain structure modelling via thousands of processors, each of minimal memory requirements, may be the better choice.

The outstanding question with regard to a hardware hypercube choice, is the extent to which there is a basic algorithmic mismatch between the hardware hypercube data flow along the edges of the hypercube and the software hypercube "subhypercubing" search process - despite the "hypercube" commonality in name.

A further possibility is to recode the Prolog system described herein into 'parallel Prolog' or "Parlog" (Conlon, 1989).

APPENDIX 5.

Getagroup.

---

1. The specific to general Hypercube Processor.

The specific to general hypercube processor is outlined in the following notes where the main modules involved are indicted in brackets, e.g. (genform - meaning 'generate the formula').

1. Hypercube processor. (main)
   - 1.1 Asks user for dimension required.
   - 1.2 Sets up the type of machine required. (setmode)
   - 1.3 Runs the hypercube for the given modes & dimension. (runcube)
   - 1.4 Asks user whether to exit or process a new dimension hypercube. In which case, it first eliminates the old hypercube from memory.

1.2 Sets up the type of machine required. (setmode)

1.2.1 Picks route from mode group ('mlos'), as input:

Manual/automatic;
Learning/total;
One/all;
Suggested/user.

Each pair represents a boolean choice (true/false or false/true).
Valid groups are (from first letters) : mlas, mtou, atau;
where:

mlas => manual initialisation and the system allows learning queries and is incremental and learning continues until the user says stop,
mtou => manual total cube and done once,
atau => for tests: auto total, all possibilities and the system is the user!

where:
Manual => values are put into the hypercube manually.
Automatic => values are put into the hypercube automatically.
Learning => the hypercube is set up with incomplete set of values.

Total => the hypercube is set up with complete set of values.
One => try only one set of hypercube possibilities.
All => try all the hypercube possibilities.
Suggested => system suggests critical indexes for which the user is then required to give the equivalent class.
User => user states critical indexes and equivalent class response.

In overview the module: setmode

1. Picks route from mode group, as input:
   Manual/automatic; Learning/total; One/all; Suggested/user.
   where valid groups are: mlas, mtou and atau.
2. and generates the input cube from the formulae and generates the formula test from some 'detectors' named prt, prt, prf, prg and prdiff.

1.3	Runs the hypercube for the given modes and dimension.	(runcube)
1.3.1	Initialises the cube.	(initcube)
1.3.2	Computes the formula for the hypercube.	(getform)
1.3.3	Generates the input cube from the formula.	(genform)
1.3.4	Compares the test & generated cubes, reporting both if different.	(comptg)

2 Initialisation.

An initialisation module allows addition of information to the hypercube by one of three operations: 'manual learn', 'auto learn' and 'manual total'. In outline these are as follows:

In 'manual learn', on the first time of entry the user inputs the indexes and class of the exemplars and the formula is produced based on interpolative (in between nodes) assumptions. On subsequent interactive entries, either the user may query the system in which case it then finds the answer from the formula generated on the previous cycle allowing the user to 'agree' or 'disagree' or not comment resulting in changes in the 'runtime mode'; or it suggests queries e.g. finds certain appropriate sets and the user answers in agreement or disagreement with the assumed formula or offers changes to the building or learning modes of the hypercube. Later the system produces the revamped formula according to whether changes were required.

In 'auto total', on the first time of entry it inputs the entire cube with all index values set to zero and later produces the formula. On subsequent interactive entries, there is a

boolean increment of the value set each time until all ones are set in cubes 0 to n and later it produces the formula.

In 'manual total', on the first and subsequent times the user inputs the entire cube and later it produces the formula.

3 Get Formula.

The module 'getform' computes the formula for the hypercube. Its arity being defined by:

1. inputs:	dimension, 'mlos', whole cube as input, initial list of lists of formulae, changes to assumed formula, number of nodes, counter (initially zero)
2. outputs:	connected group resulting from getagroup, resulting list of lists of formulae

In overview there are two module types:

1: Default:	when there are no more connected groups, but not first group tried (since it would always fail, the group being a null list on first entry.
2: Typical:	it repeatedly gets a connected group in bottom-up mode and converts to formulae.

The sub-module 'getagroup' finds a connected group and has procedures for:

1:	Entry point on exiting a group (last I/O list = null)
2:	Entry point on starting processing a group
3:	Entry point for typical processing of a group.

In operation it finds a node and associated I/O (by the getpartgroup module), accumulates the IO's and processes each node in IO group (by getagroup), until the I/O group = null list. The sub-module getapartgroup finds a node and its associated I/O as part of a connected group. Sub-modules involve: 1: the typical case, 2: default if above fails (i.e. retract fails). In operation some major components are:

assert/retract:	retracts an index which is a '1', replaces by 'x'
	=> signifies already processed and was a '1' to any later inspection.
conc:	the index is added to 'group'.
nodeio:	gets the I/O lists for index. Given the dimension and a node index, it returns the list of input node indices and the list of output node indices.
conc:	puts I/O lists into a single list.
reduceio:	reduces I/O list to only those containing '1's in cube.
elimiogroup:	eliminates from I/O list any in 'group'.

where 'group' => unique list of indices = 1 which are linked in the cube into a group and ready for converting to formulae.

4 Convert to Formula.

The purpose of this module is to convert the hypercube output to the formula format. Thus 'converttoform' has the description: convert the group to formulae format for a connected group. Its arity being defined by:

1. inputs: dimension, total number of nodes in a cube of dimension D, group resulting from getagroup, initial list of lists of formulae.
2. outputs: initial list of lists of formulae, final resulting list of lists of formulae, getform counter.

Various predicates deal with the cases:

1: Cube is all zeros.
2: General default case, when there are no more groups.
3: Cube is all ones.
4: General case, process group sublist.

In operation it connects group G to formulae and adds as list of lists of old formulae F as new formulae F1.

The sub-module: 'groupsublist' has the description: given a connected group node list, returns the list of lists. The parameters involved are:

input: dimension, groupsublist dimension counter, groupsublist dimension level counter, connected group, format: [length|Glist], list of lists of subhypercube representations of format: [[length,case,complete,list],...]], initial trial list of lists: as the complete subgroup set, but, first time through it's set as null list.
output: resulting list of lists, resulting trial list of lists.

Various predicates deal with:

1: Default for end of search of nodes in group (ie last dimension, last level)
2: Default for end of dimension for general level
3: General case when sublist < 2^{D-1}, (i.e. the "complete" subgroup is set to false)
4: General case when sublist = 2^{D-1} , (i.e. the "complete" subgroup is set to true).

In operation it searches backwards along the dimensions from Dth to 1st, then down by level (eg 1st = .,1,.,.. 2nd = .,1,.,0,. until it obtains the complete set of hypercube subgroups of

the original group, exiting when complete with equivalent list of lists of subformulae. As an example, where 'bar' after a letter implies its complement:

Abar + C -> 111,101,001,000,010,011

	ABC		
	111		=> C
(110)	101	011	
(100)	010	001	=> Abar
	000		

sublistindex	sublist			
0..	001,000,010,011	=>	2^{D-1}	=> Abar
1..	111,101	=>	2^{D-2}	
.0.	101,001,000	=>	$> 2^{D-2}$, $< 2^{D-1}$	
.1.	111,010,011	=>	$> 2^{D-2}$, $< 2^{D-1}$	
..0	000,010	=>	2^{D-2}	
..1	111,101,001,011	=>	2^{D-1}	=> C

Given the code: SI/SL => sublistindex/sublist

0. if length = 0 then no cube! ie "false" 1, .. n
1. if length = 2^D then whole cube! ie "true" 1, .. n
2. group -> SI/SL
3. ORDER SI/SL, 1, .. n by longest 1st
4. for 1, .. n
   if SI/SL(i) has length = 2^{D-1} then formula = X (or Xbar)
   else note dummy & perform recursion
   (eg .0. 101,001,000 -> 0dummy. 000, 001 1dummy. 101),

   that is: if 1 to i formulas uses all in group then exit by collecting 1 to i formulae
   else continue.

A few further notes are in order on speed up and detection:

1. If there is only one in the sublist then it does not subdivide further. On searching, it looks to see if there is only ONE and assumes any remaining deeper layers to be processed. But the algorithm assumes scan N times, unless it scans until found, i.e. picks last if < N.
2. If number of examples/nodes = n, max sublist length = N, N = n * D.
   eg N = 1000*100 = 100,000.
3. Example: limit of indexing = 13 for FULL cube set since above eg D = 15, => 32,000 nodes, if half full = 16,000, => N = 16000 * 15 = 240,000 => 1/4 MB storage/sublist = 4MB.

4. If there are > $(2^D)/2$ in one binary class, swops over 0 & 1, & re-reverses on exit. (This works well and considerably increases speed for difficult problems)
5. Limit = 6 on FULL input cube.
6. Limit = 255 on partially indexed cube.
7. If disconnected groups = D + 1 and all are size = 1 => exclusive OR.
   If disconnected groups = D and all are size = 1 => exclusive NOR.
8. Considers nearly functions.
9. Considers majority functions.
10. Given the examples, lets PROLOG ramp up from 1, until the dimensionality is found.

The sub-module 'dosublist' has the description: converts the group list to a list of sublist structures. The arity defined by:

inputs: dimension, dimension counter, level counter, group, initial list of lists of subgroups

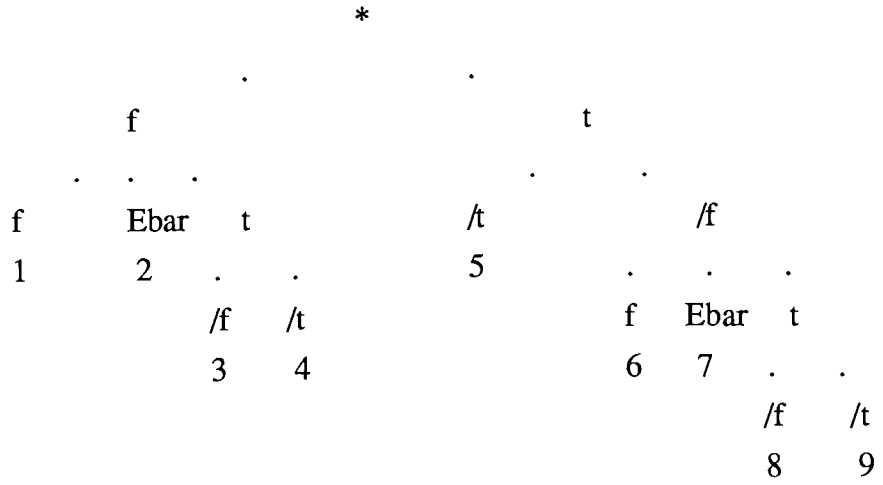
outputs: resulting list of lists of subgroups, completeness, initial trial list of lists: as the complete subgroup set, but the first time through it's set as null list, resulting trial list of lists

Some major procedures define:

1: level1: sublist = false => incomplete group
2: level1: sublist = true & in/complete group
3: levels2+ general case & sublist = false => incomplete group
4: levels2+ general case & sublist = true => in/complete group

In operation: dosublist for 1st level (dosublist's 1 to 9), where:
f = false, t = true, Ebar = does not exist, and / => result of iscomplete's

Example:-



The clauses numbering is:

1 1... = Eff
2 0... /= EfEbar
3 0... = Eft/f
4 0... = Eft/t
5 1... = Et/t
6 1... /= Et/ff
7 0... = Et/fEbar
8 1... = Et/ft/f
9 1... = Et/ft/t

The procedure: 'formcase' produces the matching D - 1 case prior to formulation.

The procedure: 'casetoform' accumulates formulas from list of lists of subgroups of the transformations of case to formula for a connected group.

The procedure: 'issubgroup' applies the case to group producing subgroup structure: [[length,case,complete,list],...]] and tests if complete (i.e. D - 1 subgroup). Parameters are:
input: dimension, case list e.g. [.,.,1,.,0,.], group,
output: case equivalent subgroup structure as: [length,Case,Complete,List], complete.

The module: 'iscomplete' checks if subgroups sum to the complete group. Parameters are:

input: connected group, format: [length|Glist], list of lists of subgroup subhypercube representations of format: [[length,case,complete,list],...]] but the first time through it's set as null list, latest subgroup list, complete set of sublists sum to group with a boolean reply true/false, initial trial list of lists: as the complete subgroup set but the first time through it's set as null list,
output: resulting trial list of lists.

In operation it first checks if latest subgroup list "complete" = true, if so, adds to trial set, if and only if not there already, (ie get sublist into trial, if and only if not a duplicate of existing list). Then checks if trial set = group, if so, it sets 'iscomplete' "complete" = true.

The procedure: 'transform' translates the subgroup to an AND'ed formula. For example: [A,.,B,.,Cbar].

APPENDIX 6.

Hypercolumn connectionism.

It is important to see the hypercube concept and its associated theory and practice in perspective. Neural nets are inspired by an attempt to build simulations and abstract computer models of actual neurons. As such they represent "**neural connectionism.**" The hypercube, however, was inspired by an attempt to build an abstract computer model of a higher neural structure, namely the hypercolumn. As such, the hypercube represents "**hypercolumn connectionism.**" The thinking behind this change, partly as a result of the 'getagroup' experience, was the suspicion that neurons themselves may not be a functionally useful point of departure. The hypercolumn seems to be a more functional, more 'computer-like' representation.

An analogy may be useful. In attempting to build aeroplanes we found that basing our technology upon feathers was not particularly insightful. Such slavish adherence to biology rarely seems to nurture new technology; yet it seems reversely so for near misses. On forgetting feathers and considering the higher level functionality of the wing as a whole, the aerofoil shape generated a workable technology. It was then seen that there are many ways of flying without resort to 'feather technology'.

Neural connectionism in its many forms has led to a 'suck it and see' technology, where, reversely, theory follows experiment. Strange effects such as 'getting stuck' follow. It is difficult if not impossible, at times, to know what the machine is doing. If we analogously equate neurons with feathers then it is perhaps better to work at a higher, more functional level. Within the brain such a level is already known to neurophysiologists, it is the hypercolumn.

The hypercolumn has some remarkable properties. From a software engineering viewpoint the hypercolumn is "well-engineered"; that is, it can be considered to possess the hallmarks of "quality software"; namely, **modularity, structure and hierarchy** (Ratcliff, 1987). Hypercolumns (in the visual cortex also referred to as "orientation columns") have the requisite high internal cohesion, low external coupling, high functionality, are very highly structured and interact hierarchically (Hubel, 1988; Rose and Dobson, 1985). In contrast there are many types of neuron varying from the above "quality software" unit to some very 'spaghetti-like' structures. Actual neural nets, on the small scale (say dozens of neurons), are very 'spaghetti-like' and axon-dendritic, (and the various local circuits such as) axon-axonic, dendritic-dendritic and cell body axonic/dendritic/cell body synaptic connections are also, in general, very 'spaghetti-like' as seen in any "Golgi trace."

A second analogy is useful. Consider the macroscopic determinism and microscopic indeterminacy of physics. Hypercolumns together with their immediate component parts seem to be functionally distinctly **deterministic**; whereas smaller neural nets and in particular their components, the synapses, appear to be functionally rather **indeterministic** - as disclosed by, for example, the weight relaxation and statistical techniques employed in their emulation.

A third analogy is also useful. Hypercolumns are composed of typically about 10^5 neurons (Hubel and Wiesel, 1979) and have components such as variations on a bar width composed of about 10^2 neurons. Neurons in turn have synapses as components. In choosing "hypercolumn connectionism" as a source of inspiration for our abstract model rather than neural connectionism we analogously hope to work with meaningful units such as sentences with words as components rather than meaningless units such as letters with pixels as components.

Recent excellent work using both KOHONEN NETS and BP to implement hypercolumn-like structures of simple perceptual groups such as bars, lines, corners, etc. has shown the need for a very considerable hierarchy of neural nets. The ensuing difficulties of construction still entailed 'jumps' or missing levels in this hierarchy (Sarkaria, 1990; Linsker, 1988; Fukushima *et al.*, 1983).

A little analysis shows why this is likely to occur. As stated above, hypercolumns in the v1 to v5 region of the cortex typically contain 10^5 neurons. Further, each of these neurons has on average $2 * 10^4$ synapses. Hence there are about 10^9 synapses per hypercolumn. The perceptual groups referred to above would, almost certainly, entail several hypercolumns and 'higher level' hypercolumns. Yet if we examine the state of the art in neural net research we see that rarely is this research considering nets greater than about 10^4 synapses, since otherwise the algorithm would become "impossibly slow." The difficulties of determining "what groups with what" (which is known as the "binding problem") together with the difficulties of, at the same time, ensuring invariance and lack of redundancy in the representation mean for present day networks:

> "... they require a prohibitively large number of units to represent even modest domains."

(Biederman *et al.*, 1991).

Clearly, given the above figures, even for *one* hypercolumn the simulation is underestimating by a factor of perhaps a million! Sarkaria, Linsker and Fukushima *et al.*, as referenced above, each cunningly used hierarchies of neural nets, yet it is difficult to avoid the conclusion that the search for "emergent properties" at the hypercolumn level by massive

simulations of neural nets may, in practice, be a lost cause. Many natural phenomena are "emergent", that is, irreducibly embedded within their respective stratum of complexity. An example is superconductivity.

There is considerable debate at present concerning emergent properties. Some neural phenomena may fall outside this possibility. [1] Even minor progress on the much simpler first level (section 3.5.7) for non/emergent properties could take decades or more. Hence the general principle of jumping the gaps of section 3.5.7 is attractive in order to work with meaningful (units as) wholes rather than statistical parts. Without enormously increasing the power of the representation in such a way our task looks hopeless. We are reminded here of the use this technique of increasing the grain size as used by DNA where division between units is specifically specified by "stop" markers.

Our case therefore rests upon a more attractive *practical* possibility, which is, to simulate an abstraction of the hypercolumn **directly**; as we attempt to do by the hypercube representation and its attendant subhypercubing algorithm.

---

1 For example, consciousness cannot be understood from the laws of physics, some new physics may be involved. However the leading lights of academia in all relavent disciplines are nicely split on this issue. Deciding between these two solutions is dependent upon our increasingly enhanced understanding (which is itself somewhat quantum-mechanical in nature) of the act of observation itself.

APPENDIX 7.

Prolog Implementation: The Subhypercubing Machine.

```
/*************************************************************
*                    University of Aston.                    *
*                                                            *
* Program:        learn                                      *
* Author:         David Ball                                 *
* Date created:   4.2.90                                     *
* Geneology:      global0.7, see for prior mods              *
* Purpose:        Hypercube machine: subhypercubing algorithm:*
* Invoke:         By typing: "learn."                        *
* Assumes:        Boolean database, gent input files         *
* Status:         Tested.                                    *
*                                                            *
*************************************************************
*                                                            *
* Mod Date:       8.2.90                                     *
* Mod Status:     working for 10D in 73secs, using binary chop.*
* Mod:            making makesupervars failure driven improves time*
*                 from 1 day to 95 secs,                     *
*                 making 3 concs in makesupervars as one conc1*
*                 speeds up 1 sec at 10D                     *
*                 making h(I,C) as single list for faster indexing*
*                 speeds up to 73 secs at 10D.               *
*                                                            *
* Mod Date:                                                  *
* Mod Status:                                                *
* Mod:                                                       *
*                                                            *
*************************************************************/


/**********    DYNAMICS    **********/

:- dynamic h/1.
:- dynamic t/4.
:- dynamic stack1/2.
:- dynamic stack2/2.
:- dynamic formula/2.
:- dynamic dimension/1.                  % store for post hcube processing
:- dynamic resolution/1.                 % store for post hcube processing
:- dynamic b_chop/1.
:- dynamic noise/1.                      % store for post hcube processing
:- dynamic top_bot/1.
:- dynamic level/1.
:- dynamic varsposition/1.
:- dynamic vold/1.
:- dynamic supervars/3.
:- dynamic s_mode/1.                     % stack mode (1 or 2)
:- dynamic debug/1.
:- dynamic vpos/1.
:- dynamic front/1.
```

```
:- dynamic middle/1.
:- dynamic rear/1.
:- dynamic listformula/1.


/**********      IMPORTS    **********/

% From a specified database file and from a specified generate t file.


/**********      EXPORTS    **********/

% To a specified output file.


/**********      STATICS          **********/

learn :-
        init_learn(D,R,Noise,TB,Debug,Listformula),
        statistics(runtime,_),
        search_levels(D,Noise,R,TB,Debug),
        statistics(runtime,[_,X]),
        nl,
        format('% Learning took ~3d sec.~n', [X]),
        nl,
        listingformula(Listformula),
        tidyup,
        told,
        tell(user).


/*************************************************************
*  Procedure:    listingformula                                *
*                                                              *
*************************************************************/

listingformula(false).
listingformula(true) :-
        write('% Formula is:'),
        nl,
        f(_,_),
        write('% For 1"s:'),
        nl,
        repeat,
        isformula(1,Reply),
        listingformula1(Reply),
        nl,
        write('% For 0"s:'),
        nl,
        repeat,
        isformula(0,Reply1),
        listingformula0(Reply1),
        nl,
        write('% For 1" & 0"s outstanding:'),
        nl,
        repeat,
```

```
        isformula(2,Reply2),
        listingformula2(Reply2),
        nl.
listingformula(true) :-
        write('% Unfortunately, no formula can be learnt from the data '),
        nl,
        write('% with the given learning parameter settings'),
        nl.

isformula(2,true) :-
        f(2,_), !.
isformula(1,true) :-
        f(1,_), !.
isformula(0,true) :-
        f(0,_), !.
isformula(_,false).

listingformula2(false).
listingformula2(true) :-                % modified to reduce to const goal depth
        retract(f(2,X)),
        write('f(2,'),
        write(X),
        write(').'),
        nl,
        !, fail.

listingformula1(false).
listingformula1(true) :-                % modified to reduce to const goal depth
        retract(f(1,X)),
        write('f(1,'),
        write(X),
        write(').'),
        nl,
        !, fail.

listingformula0(false).
listingformula0(true) :-                % modified to reduce to const goal depth
        retract(f(0,X)),
        write('f(0,'),
        write(X),
        write(').'),
        nl,
        !, fail.


/*************************************************************
*  Procedure:    init_learn                                    *
*                                                              *
*  Initialization routines: of dynamic database for learning(true/false),  *
*  checks dynamic database for hypercube existence & trivial cases,        *
*  that is, all 1's, all 0's or all x's are dealt with.        *
*  finds required problem dimension & if learning or non_learning.         *
*  Also gets Noise, Top_bot and builds initial stack1          *
*  Note: the number of goals made by this procedure is independent of D     *
*  and is approx 200 goals.                                    *
*  Note: given D, then there are 3 program variables: R, Noise & Top_bot.   *
*************************************************************/
```

```
init_learn(D,R,Noise,TB,Debug,Listformula) :-
        undo,
        retractall(f(_,_)),                % not in undo, since tidyup uses undo!
        get_dimension(D),
        assert(dimension(D)),
        build_stack1(D),
        adviseres(D),
        get_resolution(R,D),
        assert(resolution(R)),
        get_noise(D,Noise),
        assert(noise(Noise)),
        get_b_chop(BC),
        assert(b_chop(BC)),
        get_topbot(BC,TB),
        assert(s_mode(1)),
        get_debug(D,Debug),
        get_listformula(Listformula),
        assert(listformula(Listformula)),
        send_output(Outfile),
        tell(Outfile),
        nl, write('% ********** Program: learn **********'), nl, nl,
        write('% FILES used were:'), nl,
        write('% Output filename = '),
        write(Outfile), nl,
        tell(user),
        get_gent(Gent_input),
        tell(Outfile),
        write('% t input filename = '),
        write(Gent_input), nl,
        tell(user),
        get_database(Dbaseinput),
        tell(Outfile),
        write('% Database input filename = '),
        write(Dbaseinput), nl, nl,
        check_database(D,_),
        assert(varsposition(0)),
        assert(vold([])),
        assert(level(0)),
        init_output(D,R,BC,Noise,TB,Debug,Listformula).


/*************************************************************
*  Procedure:    init_output                                  *
*                                                             *
*************************************************************/

init_output(D,R,BC,Noise,Top_bot,Debug,Listformula) :-
        write('% PARAMETER settings were:'), nl,
        write('% Dimension = '),
        write(D), nl,
        write('% Resolution = '),
        write(R), nl,
        write('% Noise = '),
        write(Noise), nl,
        write('% Binary Chop = '),
        write(BC), nl,
        write('% Top bot = '),
        write(Top_bot), nl,
```

```
        write('% Debug = '),
        write(Debug), nl,
        write('% Listing formula = '),
        write(Listformula), nl,
        maxtime(D,R),                   % ELIM if not useful & approx accurate!
        nl,
        write('% SEARCHING levels:').


/*****************************************************************
*  Procedure:    undo                                             *
*                                                                 *
*****************************************************************/

undo :-                                 % precautionary, in case left from prior aborts!!
        retractall(h(_)),
        retractall(supervars(_,_,_)),   % using retractall since FAILS SAFE!!
        retractall(stack1(_,_)),
        retractall(stack2(_,_)),
        retractall(dimension(_)),
        retractall(resolution(_)),
        retractall(b_chop(_)),
        retractall(noise(_)),
        retractall(top_bot(_)),
        retractall(varsposition(_)),
        retractall(vold(_)),
        retractall(debug(_)),
        retractall(s_mode(_)),
        retractall(level(_)),
        retractall(vpos(_)),
        retractall(front(_)),
        retractall(middle(_)),
        retractall(rear(_)),
        retractall(listformula(_)),
        retractall(t(_,_,_,_)).


/*****************************************************************
*  Procedure:    get_dimension                                    *
*                                                                 *
*  Caution:      Tested up to H^1040 on present version of Quintus Prolog  *
*                and UNIX. However future versions could well revert to     *
*                a maximum list length of 112                     *
*                                                                 *
*****************************************************************/

get_dimension(D) :-
        write('Input dimension:'),
        nl,
        read(X),
        check_dimension(X,D).

check_dimension(D,D) :-
        D > 0,
        D < 1041.                               % tests up to 1040D & OK
check_dimension(_,D) :-
        write('Dimension out of range: 1 to 1040'),
        nl,
        get_dimension(D).
```

```
/*************************************************************
*  Procedure:    get_database                                 *
*************************************************************/

get_database(Dbaseinput) :-
        write('Database input filename?  CHECK IT!'),
        nl,
        read(Dbaseinput),
        see(Dbaseinput),
        repeat,
        read(Item),
        set_up_database(Item),
        seen,
        see(user).


/*************************************************************
*  Procedure:    send_output                                  *
*                                                             *
*************************************************************/

send_output(Outfile) :-
        write('Output filename?  CHECK IT!'),
        nl,
        read(Outfile).


/*************************************************************
*  Procedure:    set_up_database                              *
*                                                             *
*************************************************************/

set_up_database(end_of_file) :- !.
set_up_database(Item) :-
        asserta(Item),
        !, fail.


/*************************************************************
*  Procedure:    get_gent                                     *
*                                                             *
*************************************************************/

get_gent(Gent_input) :-
        write('Generated t input filename?  CHECK IT!'),
        nl,
        read(Gent_input),
        see(Gent_input),
        repeat,
        read(Item),
        set_up_gent(Item),
        seen,
        see(user).


/*************************************************************
*  Procedure:    set_up_gent                                  *
*                                                             *
```

```
**************************************************************/
set_up_gent(end_of_file) :- !.
set_up_gent(Item) :-
        assertz(Item),
        !, fail.


/**************************************************************
*  Procedure:    build_stack1                                  *
*                                                              *
*  Notes:        New faster algorithm uses incmember cf build_v *
*                D = 3, 13 cf 28 goals; D = 13, 43 cf 163 steps! *
*                                                              *
**************************************************************/

build_stack1(D) :-
        length(L,D),
        copy_term(L,L1),
        incmember(0,L1),                    % create Vposition list
        assert(stack1(L1,L)).

incmember(X,[X]).
incmember(X,[X|R]) :-
        X1 is X + 1,
        incmember(X1,R).


/**************************************************************
*  Procedure:    adviseres                                     *
*                                                              *
**************************************************************/

adviseres(D) :-                             % D > 1000
        D > 1000,
        write('As a guide, for a learning time of less than one day,'),
        nl,
        write('then an advisable value for resolution is 1'),
        nl,
        nl.
adviseres(D) :-                             % D > 500
        D > 500,
        write('As a guide, for a learning time of less than one day,'),
        nl,
        write('then an advisable value for resolution is 2 or less'),
        nl,
        nl.
adviseres(D) :-                             % D > 29
        D > 29,
        write('As a guide, for a learning time of less than one day,'),
        nl,
        write('then an advisable value for resolution is 3 or less'),
        nl,
        nl.
adviseres(D) :-                             % 30 > D > 15
        D > 15,
        write('As a guide, for a learning time of less than one day,'),
        nl,
        write('then an advisable value for resolution is '),
        R is 33 - D,
```

```
        write(R),
        write(', or less'),
        nl,
        nl.
adviseres(_).                                   % D < 1


/*************************************************************
*  Procedure:    get_resolution                                *
*                                                              *
*************************************************************/

get_resolution(R,D) :-
        write('Input resolution:'),
        nl,
        read(X),
        check_resolution(X,R,D).

check_resolution(R,R,D) :-
        R > 0,
        R =< D.
check_resolution(_,R,D) :-
        write('Resolution out of range: 1 to '),
        write(D),
        write(' (the Dimension).'),
        nl,
        get_resolution(R,D).


/*************************************************************
*  Procedure:    get_b_chop                                    *
*                                                              *
*  Notes:        get binary chop requirement as true or false  *
*                                                              *
*************************************************************/

get_b_chop(BC) :-
        write('Input binary chop:'),
        nl,
        read(X),
        check_b_chop(X,BC).

check_b_chop(y,true).
check_b_chop(n,false).
check_b_chop(_,BC) :-
        write('Binary chop must be stated as `y` for true or `n` for false'),
        nl,
        get_b_chop(BC).


/*************************************************************
*  Procedure:    maxtime                                       *
*                                                              *
*  Note:         if % linearity is used => could produce estimated *
*                time, cf max (parity only) time.              *
*                                                              *
*************************************************************/

maxtime(D,R) :-
```

```
        D > 10,
        R < 5,
        top_bot(Top_bot),
        get_time(D,R,Top_bot,T),
        nl,
        write('% Estimated maximum processing time is '),
        write(T),
        write(', minutes'),
        nl.
maxtime(_,_).

get_time(D,1,true,T) :-
        T is 0.05 * D / 60.
get_time(D,1,false,T) :-
        T is 0.1 * D / 60.
get_time(D,2,true,T) :-
        T is 0.05 * D * D / 60.
get_time(D,2,false,T) :-
        T is 0.1 * D * D / 60.
get_time(D,3,true,T) :-
        T is 0.05 * D * D * D / 60.
get_time(D,3,false,T) :-
        T is 0.1 * D * D  * D / 60.
get_time(D,4,true,T) :-
        T is 0.05 * D * D * D * D / 60.
get_time(D,4,false,T) :-
        T is 0.1 * D * D * D * D / 60.


/*************************************************************
*  Procedure:    get_noise                                     *
*                                                              *
*************************************************************/

get_noise(D,Noise) :-
        write('Input noise:'),
        nl,
        read(Input),
        check_noise(D,Input,Noise).

check_noise(D,1,1) :- D >= 1.
check_noise(D,2,2) :- D >= 2.
check_noise(D,3,3) :- D >= 3.
check_noise(D,4,4) :- D >= 4.
check_noise(D,Input,Noise) :-
        write('Noise = '),
        write(Input),
        write(', is either: out of the range 1 to 4, '),
        write('or > dimension  '),
        write(D),
        nl,
        nl,
        write('Input the noise tolorance where: '),
        nl,
        write('   1 => database is assumed to have no noise'),
        nl,
        write('   2 to 4 => correspondingly increasing noise levels'),
        nl,
        get_noise(D,Noise).
```

```
/*************************************************************
*  Procedure:    get_topbot                                    *
*                                                              *
*************************************************************/

get_topbot(true,false) :-
        write('NOTE: Binary Chop is true, so Top Bot has to be false'),
        nl.
get_topbot(false,Topbot) :-
        write('Top bot required?'),
        nl,
        read(Input),
        check_topbot(Input,Topbot).

check_topbot(y,true).
check_topbot(n,false).
check_topbot(_,Topbot) :-
        write('Incorrect response'),
        nl,
        write('Top bot may be invoked for fast approximation'),
        nl,
        write('Input y to invoke top bot or n if not required'),
        nl,
        get_topbot(false,Topbot).


/*************************************************************
*  Procedure:    get_listformula                               *
*                                                              *
*************************************************************/

get_listformula(Listformula) :-
        write('List formula required?'),
        nl,
        read(Input),
        check_listformula(Input,Listformula).

check_listformula(y,true).
check_listformula(n,false).
check_listformula(_,Listformula) :-
        write('Incorrect response'),
        nl,
        write('Input y to list formula to database or n if not required'),
        nl,
        get_listformula(Listformula).


/*************************************************************
*  Procedure:    check_database                                *
*                                                              *
*  Format:       of database is "h([Class| Index])."           *
*************************************************************/

check_database(D,Cube) :-
        length(Cube,D),
        testdatabase(Cube).

testdatabase(Index) :-
```

```
        testdata1(Index,1,Ans1),
        testdata0(Index,0,Ans0),
        testdatax(Index,x,Ansx),
        testans(Index,Ans1,Ans0,Ansx).

testdata1(Index,1,true) :-
        copy_term(Index,Index1),
        h([1|Index1]).
testdata1(_,_,false).

testdata0(Index,0,true) :-
        copy_term(Index,Index0),
        h([0|Index0]).
testdata0(_,_,false).

testdatax(Index,x,true) :-
        copy_term(Index,Indexx),
        h([x|Indexx]).
testdatax(_,_,false).

testans(Index,true,false,false) :-                    % database all 1's
        assertz(f(1,Index)),
        write('% NOTE: hypercube is all 1"s!!'), nl,
        write('% Formula is: '), write(Index), nl,
        abort.
testans(_,false,true,false) :-                        % database all 0's
        write('% NOTE: hypercube is all 0"s, '),
        write('so there is NO formula!!'), nl,
        abort.
testans(_,true,true,false).                           % database all 1,0
testans(_,_,_,true) :-                                % database has x's
        write('% NOTE: hypercube has x"s, '),
        write('so data base incorrectly set up!!'), nl,
        abort.
testans(_,_,_,_) :-                                   % database has ??
        write('% NOTE: data base has been incorrectly set up '),
        write('or not set up at all!!'), nl,
        abort.


/*************************************************************
*  Procedure:    get_debug                                     *
*                                                              *
*************************************************************/

get_debug(D,Debug) :-
        write('Debug required?'),
        nl,
        read(Input),
        check_debug(D,Input,Debug).
get_debug(_,_).

check_debug(_,y,true).
check_debug(_,n,false).
check_debug(D,_,Debug) :-
        write('Incorrect response'),
        nl,
        write('If debug is required then stacked subhypercubes '),
        nl,
```

```
        write('are printed out for inspection as they are stacked,'),
        nl,
        write('and elected halving by top_bot, if used is stated.'),
        nl,
        write('But note that if D > 10, then you are best avoiding'),
        nl,
        write('using debug, since the output would become excessive!!'),
        nl,
        write('Input y if debug is required, else n'),
        nl,
        get_debug(D,Debug).


/*************************************************************
*  Procedure:    tidyup                                        *
*                                                              *
*  Tidies up dynamic database & outputs the formula before program exits.  *
*                                                              *
*************************************************************/

tidyup :-
        undo,
        exitglobal1.                          % iff non-incremental!

exitglobal1 :-
        repeat,
        testcasex(1,_,Ans),                   % whatever the Noise level!
        exitglobal(Ans).

exitglobal(false).
exitglobal(true) :-
        retract(h([x|L])),
        assertz(h([1|L])),
        !, fail.


/*************************************************************
*  Procedure:       search_levels                              *
*                                                              *
*  Description:   "repeat ... !, fail." format is used throughout from  *
*                   now on, and relies on throwing away the clause upon  *
*                   reaching fail, & backtracking to repeat. Alternative  *
*                   clauses succeed, thus do NOT backtrack to repeat but  *
*                   continue with next clause. eg if levelsearch succeeds  *
*                   then go ON to "repeat ... make_formula" loop.  *
*                                                              *
*************************************************************/

search_levels(D,Noise,Limit,TB,Debug) :-      % search hcube
        repeat,                               % search levels loop
        level(Level),
        s_mode(Mode),
        is_stack(Mode,V,Stack1reply),
        levelsearch(D,Noise,Limit,Level,TB,Debug,Mode,V,Stack1reply).

is_stack(1,V,true) :-
        stack1(V,_), !.
is_stack(2,V,true) :-
        stack2(V,_), !.
```

```
is_stack(_,_,false).

/*************************************************************
*  Procedure:    levelsearch                                  *
*                                                             *
*  Purpose:      To search along a given level, (replaces 1st, 2ndmodes)  *
*                                                             *
*************************************************************/

levelsearch(_,_,_,_,_,_,_,_,false) :- !.              % no stack1 left
levelsearch(D,1,D,Level,_,_,_,_,_) :-                 % at node level
        Level is D - 1,
        listformula(Ans),
        start_search(Level),
        length(L,D),
        repeat,
        copy_term(L,L1),
        testcase1(1,L1,Reply),
        h_to_formula(L,Reply,Ans),
        !.
levelsearch(D,Noise,_,Level,_,_,Mode,_,_) :-          % noise limit
        Level is D - Noise + 1,
        start_search(Level),
        length(L,D),
        repeat,
        teststack(Mode,Reply),
        stack_to_formula(Mode,L,Noise,Reply), !.      % straight convert 1 - 4D's
levelsearch(D,Noise,Limit,Limit,_,_,Mode,_,_) :-      % resolution limit
        start_search(Limit),
        length(L,D),
        repeat,
        teststack(Mode,Reply),
        stack_to_formula(Mode,L,Noise,Reply), !.
levelsearch(D,Noise,_,Level,TB,Debug,Mode,V,true) :-  % general case
        start_search(Level),
        length(V,Vlength),
        length([X|R],Vlength),
        assert(vpos(0)),
        assert(front([])),
        assert(middle([X])),
        assert(rear(R)),
        repeat,
        rear(Rear),
        makesupervars(Rear,Vlength),
        repeat,                                       % firstmode loop
        get_stack(Mode,Vold,Iold,Stackreply),         % see if more stack
        testsubhcube(D,Noise,TB,Debug,Mode,Vold,Iold,Stackreply),
        end_search,
        !, fail.

h_to_formula(_,false,_).
h_to_formula(L,true,Ans) :-          % modified to reduce to const goal depth
        retract(h([1|L])),
        assertz(h([x|L])),
        make_formula(1,L,Ans),
        !, fail.

make_formula(_,_,false).
make_formula(Type,Index,true) :-
```

```
        assertz(f(Type,Index)).

start_search(Level) :-                          % ALWAYS O/P since so useful!
        nl,
        write('% Generating level '),
        write(Level),
        nl.

teststack(1,true) :-
        stack1(_,_), !.
teststack(2,true) :-
        stack2(_,_), !.
teststack(_,false).

stack_to_formula(1,L,Noise,true) :-             % now accounts for noise
        copy_term(L,L1),
        retract(stack1(_,L1)),
        testcases(Noise,L1,[Case1,Case0,Casex]),
        testformula(L1,[Case1,Case0,Casex]),
        !, fail.                                % modified to reduce to const goal depth
stack_to_formula(2,L,Noise,true) :-             % now accounts for noise
        copy_term(L,L1),
        retract(stack2(_,L1)),
        testcases(Noise,L1,[Case1,Case0,Casex]),
        testformula(L1,[Case1,Case0,Casex]),
        !, fail.
stack_to_formula(_,_,_,false).

testcases(Noise,Top,[Case1,Case0,Casex]) :-
        copy_term(Top,Top1),                    % outside to save backtracking
        testcase1(Noise,Top1,Case1),
        copy_term(Top,Top0),                    % outside to save backtracking
        testcase0(Noise,Top0,Case0),
        copy_term(Top,Topx),                    % outside to save backtracking
        testcasex(Noise,Topx,Casex).

testformula(L,[true,true,_]) :-                 % 1's & 0's
        listformula(Ans),
        make_formula(2,L,Ans), !.
testformula(L,[false,true,false]) :-            % all 0's only
        listformula(Ans),
        make_formula(0,L,Ans), !.
testformula(L,[true,false,_]) :-                % all 1's or all 1's & x's
        listformula(Ans),
        make_formula(1,L,Ans),
        repeat,
        copy_term(L,L1),
        testcase1(1,L1,Reply),
        convertclass(L1,Reply).
testformula(_,_).

get_stack(1,V,I,true) :-
        retract(stack1(V,I)), !.
get_stack(2,V,I,true) :-
        retract(stack2(V,I)), !.
get_stack(_,_,_,false).

end_search :-
        retract(level(Level)),
```

```
        Level1 is Level + 1,
        assert(level(Level1)),
        retractall(supervars(_,_,_)),
        retract(s_mode(Mode)),
        flipmode(Mode,Mode1),
        assert(s_mode(Mode1)).

flipmode(1,2).
flipmode(2,1).


/*************************************************************
*  Procedure:      makesupervars                               *
*                                                              *
*  Purpose:        To avoid duplicate subhypercubes being stored!!  *
*                                                              *
*  Description:    makesupervars is a machine to generate appropriate  *
*                  "supervars" templates for a given level - thus  *
*                  infrequently used, ie only once per level! supervars  *
*                  are a fast database template for how to elim ONE bool  *
*                  from list & combine rest into the new list. This is  *
*                  achieved in just ONE goal, so avoiding expensive concs  *
*                  in "expand".                                *
*                  supervars format: supervars(Vposition,Vold,Vnew).  *
*                   eg supervars(1,[1,0,1,0],[1,1,0])          *
*                  Note: when Vnew = [] => no more vars left ie all bools!  *
*                                                              *
*************************************************************/

makesupervars([],Vlength) :-                    % final template
        length(L,Vlength),
        retract(vpos(Vposition)),
        retract(front(Front)),
        retract(middle(Middle)),
        retract(rear(_)),
        conc(Front,Middle,L),
        assertz(supervars(Vposition,L,Front)).
makesupervars(_,Vlength) :-                     % build general template
        length(Vold,Vlength),
        retract(vpos(Vposition)),
        retract(front(Front)),
        retract(middle(Middle)),
        retract(rear([X|R])),
        conc1(Front,Middle,[X|R],Vold,Vnew,Front1),
        assertz(supervars(Vposition,Vold,Vnew)),
        Vposition1 is Vposition + 1,
        assert(vpos(Vposition1)),
        assert(front(Front1)),
        assert(middle([X])),
        assert(rear(R)),
        !, fail.

conc([],L,L).
conc([X|L1],L2,[X|L3]) :-
conc(L1,L2,L3).

conc1([],[],L,L,L,[]).
conc1([X|L1],L2,L3,[X|L4],[X|L5],[X|L6]) :-
conc1(L1,L2,L3,L4,L5,L6).
```

```
conc1([],[X],L3,[X|L4],L5,[X]) :-
conc1([],[],L3,L4,L5,[]).


/*************************************************************
*  Procedure:    testsubhcube                                  *
*                                                              *
*  Purpose:      To test the subhypercube                      *
*                                                              *
*************************************************************/

testsubhcube(_,_,_,_,_,_,_,false) :- !.
testsubhcube(D,Noise,TB,Debug,Mode,Vold,Iold,true) :-
        copy_term(Iold,Iold1),                  % outside to save backtracking
        testcase1(Noise,Iold1,Case1),
        copy_term(Iold,Iold0),                  % outside to save backtracking
        testcase0(Noise,Iold0,Case0),
        runcase(D,Noise,TB,Debug,Mode,Vold,Iold,[Case1,Case0]),
        !, fail.

testcase1(1,Top,true) :-                        % 2 goals for T
        h([1|Top]), !.
testcase1(2,Top,Reply) :-                       % max 6 goals for T/F
        copy_term(Top,Top1),
        retract(h([1|Top])),
        testcase1(1,Top1,Reply),
        assertz(h([1|Top])), !.
testcase1(3,Top,Reply) :-                       % max 10 goals for T/F
        copy_term(Top,Top1),
        retract(h([1|Top])),
        testcase1(2,Top1,Reply),
        assertz(h([1|Top])), !.
testcase1(4,Top,Reply) :-                       % max 14 goals for T/F
        copy_term(Top,Top1),
        retract(h([1|Top])),
        testcase1(3,Top1,Reply),
        assertz(h([1|Top])), !.
testcase1(_,_,false).                           % min 3 to 5 goals for F

testcase0(1,Top,true) :-                        % 2 goals for T
        h([0|Top]), !.
testcase0(2,Top,Reply) :-                       % max 6 goals for T/F
        copy_term(Top,Top1),
        retract(h([0|Top])),
        testcase0(1,Top1,Reply),
        assertz(h([0|Top])), !.
testcase0(3,Top,Reply) :-                       % max 10 goals for T/F
        copy_term(Top,Top1),
        retract(h([0|Top])),
        testcase0(2,Top1,Reply),
        assertz(h([0|Top])), !.
testcase0(4,Top,Reply) :-                       % max 14 goals for T/F
        copy_term(Top,Top1),
        retract(h([0|Top])),
        testcase0(3,Top1,Reply),
        assertz(h([0|Top])), !.
testcase0(_,_,false).                           % min 3 to 5 goals for F
```

```
/*************************************************************
*  Procedure:    runcase1                                      *
*                                                              *
*************************************************************/

%       pushes stack as appropriate.

runcase(_,_,_,_,_,_,_,_,[false,false]).          % none or all: (0's & x's), x's
runcase(_,Noise,_,_,_,_,Top,[false,true]) :-     % all 0's ensure no x's!!
        copy_term(Top,Top1),
        testcasex(Noise,Top1,Casex),
        formula0(Top,Casex).
runcase(_,_,_,_,_,_,Top,[true,false]) :-         % (1's & x's) or all 1's
        listformula(Ans),
        make_formula(1,Top,Ans),
        repeat,
        copy_term(Top,Top1),
        testcase1(1,Top1,Reply),
        convertclass(Top1,Reply).
runcase(D,_,TB,Debug,Mode,Vold,Iold,[true,true]) :-      % (1,0,x's) or (1,0's)
        retract(varsposition(_)),
        assert(varsposition(0)),
        retract(vold(_)),
        assert(vold(Vold)),
        get_top_bot(TB,[Top,Bot],Iold),
        ondebug(Debug,TB,Top,Bot),
        repeat,                                  % create subhcubes loop
        vold(Vold1),
        expand(D,[Top,Bot],Debug,Mode,Vold,Iold,Vold1).  % create subhcubes

testcasex(1,Top,true) :-                         % 2 goals for T
        h([x|Top]), !.
testcasex(2,Top,Reply) :-                        % max 6 goals for T/F
        copy_term(Top,Top1),
        retract(h([x|Top])),
        testcasex(1,Top1,Reply),
        assertz(h([x|Top])), !.
testcasex(3,Top,Reply) :-                        % max 10 goals for T/F
        copy_term(Top,Top1),
        retract(h([x|Top])),
        testcasex(2,Top1,Reply),
        assertz(h([x|Top])), !.
testcasex(4,Top,Reply) :-                        % max 14 goals for T/F
        copy_term(Top,Top1),
        retract(h([x|Top])),
        testcasex(3,Top1,Reply),
        assertz(h([x|Top])), !.
testcasex(_,_,false).                            % min 3 to 5 goals for F

formula0(Top,false) :-
        listformula(Ans),
        make_formula(0,Top,Ans), !.
formula0(_,true).

convertclass(_,false) :- !.
convertclass(Top,true) :-                        % convert all 1's to x's.
        retract(h([1|Top])),
        asserta(h([x|Top])),
```

```
        !, fail.

ondebug(false,_,_,_).
ondebug(true,false,_,_).
ondebug(true,true,false,false).
ondebug(true,true,false,true) :-
        write('    Base = true'),
        nl.
ondebug(true,true,true,false) :-
        write('    Top = true'),
        nl.
ondebug(true,true,true,true) :-
        write('    Top = true'),
        nl.


/*************************************************************
*  Procedure:    expand                                       *
*                                                             *
*  Purpose:      To expand the subhypercube                   *
*                                                             *
*************************************************************/

expand(_,_,_,_,_,_,[]) :- !.                          % end creating subhcubes
expand(D,[Top,Bot],Debug,Mode,Vold,Iold,[X|R]) :-     % create subhcubes
        retract(varsposition(N)),                     % get next vposition index
        supervars(N,Vold,Vnew),                       % reduce Vold to Vnew
        t(X,Ones,Zeros,D),              % MOD FOR t MACHINE! one/zero template db
        expandtop([Top,Bot],Debug,Mode,Vnew,Iold,Ones),     % expand top of suhcube
        expandbase([Top,Bot],Debug,Mode,Vnew,Iold,Zeros),   % expand bot suhcube
        N1 is N + 1,                                  % inc Vposition counter
        assert(varsposition(N1)),
        retract(vold(_)),
        b_chop(BC),
        binary_chop(BC,R,Rest),
        assert(vold(Rest)),                           % get ready to expand again
        !, fail.

expandtop([true,_],Debug,Mode,Vnew,Iold,Ones):-       % expand top of suhcube
        copy_term(Iold,Inew1),
        Inew1 = Ones,                                 % one unify new subhcube
        duplicate_test(Debug,Mode,Vnew,Inew1).        % test new1 subhcube
expandtop([false,_],_,_,_,_,_).

expandbase([_,true],Debug,Mode,Vnew,Iold,Zeros):-     % expand top of suhcube
        copy_term(Iold,Inew0),
        Inew0 = Zeros,                                % Zero unify new subhcube
        duplicate_test(Debug,Mode,Vnew,Inew0).        % test new0 subhcube
expandbase([_,false],_,_,_,_,_).

duplicate_test(_,1,V,I):-
        stack2(V,I), !.
duplicate_test(_,2,V,I):-
        stack1(V,I), !.
duplicate_test(Debug,1,V,I):-
        assertz(stack2(V,I)),                         % opposite mode!!!
        ifstack(Debug,1,V,I), !.
duplicate_test(Debug,2,V,I):-
        assertz(stack1(V,I)),                         % opposite mode!!!
```

```
        ifstack(Debug,2,V,I).

ifstack(false,_,_,_) :- !.
ifstack(true,1,V,I) :-
        write('% stack1('),
        write(V),
        write(','),
        write(I),
        write(')'),
        nl, !.
ifstack(true,2,V,I) :-
        write('% stack2('),
        write(V),
        write(','),
        write(I),
        write(')'),
        nl.

binary_chop(false,R,R).
binary_chop(true,_,[]).


/*****************************************************************
*  Procedure:    get_top_bot                                      *
*                                                                 *
*  Comment:      If top_bot is false, then the init_learn assertions of Top   *
*                and Base (both true) will be used throughout,    *
*                otherwise Top and Base will be altered for each new   *
*                expansion accordingly.                           *
*                                                                 *
*****************************************************************/

get_top_bot(true,[Top,Bot],Topstack) :-
        copy_term(Topstack,Topstack1),
        converttobase(Topstack1),
        testtopbase(Topstack1,Bot),
        trytop(Topstack,Top,Bot), !.
get_top_bot(false,[true,true],_).

converttobase([0|[]]).                          % slick side effect, vars -> 0
converttobase([1|[]]).

converttobase([0,0|[]]).
converttobase([1,0|[]]).
converttobase([0,1|[]]).
converttobase([1,1|[]]).

converttobase([0,0,0|[]]).
converttobase([1,0,0|[]]).
converttobase([0,1,0|[]]).
converttobase([1,1,0|[]]).
converttobase([0,0,1|[]]).
converttobase([1,0,1|[]]).
converttobase([0,1,1|[]]).
converttobase([1,1,1|[]]).

converttobase([0,0,0|R]) :-
        converttobase(R).
converttobase([1,0,0|R]) :-
```

```
        converttobase(R).
converttobase([0,1,0|R]) :-
        converttobase(R).
converttobase([1,1,0|R]) :-
        converttobase(R).
converttobase([0,0,1|R]) :-
        converttobase(R).
converttobase([1,0,1|R]) :-
        converttobase(R).
converttobase([0,1,1|R]) :-
        converttobase(R).
converttobase([1,1,1|R]) :-
        converttobase(R).

converttobase([0,0|R]) :-
        converttobase(R).
converttobase([1,0|R]) :-
        converttobase(R).
converttobase([0,1|R]) :-
        converttobase(R).
converttobase([1,1|R]) :-
        converttobase(R).

testtopbase(Index,false) :-
        h([0|Index]).
testtopbase(_,true).

trytop(_,true,false).
trytop(Topstack,Top,true) :-
        copy_term(Topstack,Topstack1),
        converttotop(Topstack1),
        testtopbase(Topstack1,Top).

converttotop([1|[]]).                    % slick side effect, vars -> 1
converttotop([0|[]]).

converttotop([1,1|[]]).
converttotop([0,1|[]]).
converttotop([1,0|[]]).
converttotop([0,0|[]]).

converttotop([1,1,1|[]]).
converttotop([0,1,1|[]]).
converttotop([1,0,1|[]]).
converttotop([0,0,1|[]]).
converttotop([1,1,0|[]]).
converttotop([0,1,0|[]]).
converttotop([1,0,0|[]]).
converttotop([0,0,0|[]]).

converttotop([1,1,1|R]) :-
        converttotop(R).
converttotop([0,1,1|R]) :-
        converttotop(R).
converttotop([1,0,1|R]) :-
        converttotop(R).
converttotop([0,0,1|R]) :-
        converttotop(R).
converttotop([1,1,0|R]) :-
```

```
        converttotop(R).
converttotop([0,1,0|R]) :-
        converttotop(R).
converttotop([1,0,0|R]) :-
        converttotop(R).
converttotop([0,0,0|R]) :-
        converttotop(R).

converttotop([1,1|R]) :-
        converttotop(R).
converttotop([0,1|R]) :-
        converttotop(R).
converttotop([1,0|R]) :-
        converttotop(R).
converttotop([0,0|R]) :-
        converttotop(R).
converttotop([0,0|R]) :-
        converttotop(R).

autogenerate :-
        startautogen,
        getalldim(All),
        getsingledim(All,D),
        rungenerate(All,D),
        endautogen.

startautogen :-
        write('AUTO-GENERATE.'),
        nl.

getalldim(Reply) :-
        write('Test all dimensions 1-4 inclusive? '),
        write('Answer y or n.'),
        nl,
        read(X),
        checkall(X,Reply).

checkall(y,true).
checkall(n,false).
checkall(_,Reply) :-
        write('Incorrect response.'),
        nl,
        getalldim(Reply).

getsingledim(true,_).
getsingledim(false,D) :-
        write('OK, testing specific dimensions.'),
        nl,
        write('Which dimension out of 1 to 4 inclusive?'),
        nl,
        read(X),
        checkdim(X,D).

checkdim(1,1).
checkdim(2,2).
checkdim(3,3).
checkdim(4,4).
checkdim(_,Reply) :-
        write('Incorrect response.'),
```

```
        nl,
        getsingledim(false,Reply).

rungenerate(false,D) :-
        generateall(D).
rungenerate(true,_) :-
        generateall(1),
        generateall(2),
        generateall(3),
        generateall(4).

endautogen :-
        write('Completed autogenerate.'),
        nl,
        nl.

autoruntime(D,N,Maxtests) :-
        startautorun,
        doautoruntime(D,N,Maxtests),
        endautorun(D,Maxtests).

startautorun :-
        write('AUTORUNTIME'),
        nl.

endautorun(D,Maxtests) :-
        write('Completed autoruntime of dimension '),
        write(D),
        write(', for '),
        write(Maxtests),
        write(' tests'),
        nl,
        nl.

doautoruntime(_,N,N).
doautoruntime(D,N,Maxtests) :-
        M is N + 1,
        write('Trial number = '),
        write(M),
        randindex(D,[],Index),
        write('Index = '),
        write(Index),
        nl,
        useformula(Index),
        nl,
doautoruntime(D,M,Maxtests).

checkhh1 :-
        starthh1,
        hh1,
        leftovers,
        restoreh,
        retractall(h1),
        endhh1.

starthh1 :-
        write('CHECK H H1. '),
        write('OK, only if no differences are reported.'),
        nl.
```

```
hh1 :-
        retract(h(A,B)),
        asserta(h(A,B,seen)),
        retract(h1(C,D)),
        asserta(h1(C,D,seen)),
        comparehh1(A,B,C,D),
        hh1.
hh1.

comparehh1(A,B,C,D) :-
        compareclasses(A,B,C,D),
        compareindexes(A,B,C,D).

compareclasses(_,B,_,D) :-
        B == D.
compareclasses(A,B,C,D) :-
        write('Compare class difference is: '),
        hform(h,A,B),
        write(', '),
        hform(h1,C,D),
        write('.'),
        nl.

hform(H,X,Y) :-
        write(H),
        write('('),
        write(X),
        write(','),
        write(Y),
        write(')').

compareindexes(A,B,C,D) :-
        comparelists(A,B,C,D),
        comparelengths(A,B,C,D),
        comparevalues(A,B,C,D).

comparelists(A,_,C,_) :-
        isalist(A),
        isalist(C).
comparelists(A,B,C,D) :-
        write('Compare indexes: list difference: '),
        hform(h,A,B),
        write(', '),
        hform(h1,C,D),
        write('.'),
        nl.

isalist([]).
isalist([_|[]]).
isalist([_|_]).

comparelengths(A,_,C,_) :-
        length(A,X),
        length(C,Y),
        X = Y.
comparelengths(A,B,C,D) :-
        write('Compare indexes: length difference: '),
        hform(h,A,B),
```

```
        write(', '),
        hform(h1,C,D),
        write('.'),
        nl.

comparevalues([],_,[],_).
comparevalues([X|R],_,[Y|S],_) :-
        X == Y,
        comparevalues(R,_,S,_).
comparevalues(A,B,C,D) :-
        write('Compare indexes: values different: '),
        hform(h,A,B),
        write(', '),
        hform(h1,C,D),
        write('.'),
        nl.

leftovers :-
        leftoverh,
        leftoverh1.

leftoverh :-
        retract(h(A,B)),
        asserta(h(A,B,seen)),
        write('Leftover difference in h: '),
        hform(h,A,B),
        write('.'),
        nl,
        leftoverh.
leftoverh.

leftoverh1 :-
        retract(h1(C,D)),
        asserta(h(C,D,seen)),
        write('Leftover difference in h1: '),
        hform(h1,C,D),
        write('.'),
        nl,
        leftoverh1.
leftoverh1.

restoreh :-
        retract(h(A,B,seen)),
        asserta(h(A,B)),
        restoreh.
restoreh.

endhh1 :-
        write('Completed check h h1.'),
        nl.

displayAvlist(Avlist) :-
        startdisplay,
        runheader,
        rundisplay(Avlist),
        nl.

startdisplay :-
        nl,
```

```
        write('Display of the input attribute/values.'),
        nl,
        nl.

runheader :-
        write('ATTRIBUTES:'),
        tab(6),
        write('VALUES:'),
        nl.

rundisplay([]).
rundisplay([X|L]) :-
        outputAvline(X),
        nl,
        rundisplay(L).

outputAvline([]) :-
        nl.
outputAvline([A|V]) :-
        outputA(A),
        outputv(V).

outputA(A) :-
        write(A),
        name(A,L),
        length(L,N),
        X is 17 - N,
        tab(X).

outputv([]).
outputv([X|L]) :-
        write(X),
        tab(2),
        outputv(L).


% 2. Display the attribute value list for the resulting equation.

displayeqAvlist(Avlist) :-
        starteqdisplay,
        runheader,
        rundisplay(Avlist),
        nl.

starteqdisplay :-
        nl,
        write('Display of the resulting equation attribute/values.'),
        nl,
        nl.

% 3. Display the attribute value list for the resulting redundancy.

displayredundAvlist(Avlist) :-
        startredunddisplay,
        runheader,
        rundisplay(Avlist),
        nl.

startredunddisplay :-
```

```
        nl,
        write('Display the resulting redundancy attribute/values.'),
        nl,
        nl.

divlist(L,Div,Lists1) :-
        length(L,N),
        shortlist is N / Div,
        buildshortlists(shortlist,shortlist,L,Div,[],_,_,Lists1).

buildshortlists(shortlist,0,[],0,Newlist,Newlist,Lists,Lists1) :-
        conc(Newlist,Lists,Lists1).
buildshortlists(shortlist,0,L,Div,Newlist,Newlist,Lists,Lists2) :-
        conc(Newlist,Lists,Lists1),
        Div1 is Div - 1,
        buildshortlists(shortlist,0,L,Div,Newlist,Newlist,Lists,Lists2).
buildshortlists(shortlist,shortlist[X|R],Div,Newlist,Newlist2,Lists,List1) :-
        conc([X],Newlist,Newlist1),
        shortlist1 is shortlist - 1,
        buildshortlists(shortlist,shortlist1,R,Div,Newlist1,Newlist2,Lists,Lists).

convert_dec_bool(Dec,Bool,Maxbi) :-
        get_lsbits(Dec,Bi),
        filltomax(Bi,Maxbi,Bool).

get_lsbits(0,[0]).
get_lsbits(1,[1]).
get_lsbits(Dec,Bi) :-
        get_bi_list(Dec,Bi).

get_bi_list(Dec,Bi) :-
        decbool(Dec,2,[1],[0],Lastbi,R),
        find_remainder(R,Lastbi,Bi).

decbool(0,_,_,_,[0],0).
decbool(1,_,_,_,[1],0).
decbool(Dec,Double,Bi,_,Lastbi,R) :-                % divider too small
        Dec > Double,
        Double1 is Double * 2,
        conc(Bi,[0],Bi1),
        decbool(Dec,Double1,Bi1,Bi,Lastbi,R).
decbool(Dec,Double,Bi,_,Bi,0) :-                    % exact division
        Dec is Double.
decbool(Dec,Double,_,Lastbi,Lastbi,R) :-  % prior division plus remainder
        Dec < Double,
        R is Dec - Double.

filltomax(Bi,Maxbi,Bool) :-
        length(Bi,N),
        N1 is Maxbi - N,
        length(L,N1),
        makezero(L,Zero),
        conc(Zero,Bi,Bool).
makezero([],[]).
makezero([X|R],[0|S]) :-
        makezero(R,S).

find_remainder(0,bi,Bi).
find_remainder(Dec,Oldbi,Bewbi) :-
```

```
        decbool(Dec,2,[1],[0],Lastbi,R),
        makenewbi(Oldbi,Lastbi,Newbi1),
        find_remainder(R,Newbi1,Newbi2).

makenewbi(Oldbi,Lastbi,Newbi1) :-
        length(Oldbi,N),
        filltomax(Lastbi,N,Lastbi1),
        addbi(Oldbi,Lastbi,Newbi).

addbi([],[]).
addbi([1|R],[0|S],[1|T]) :-
        addbi(R,S,T).
addbi([0|R],[1|S],[1|T]) :-
        addbi(R,S,T).
addbi(0|R],[0|S],[0|T]) :-
        addbi(R,S,T).

convert_bool_dec(Bool,Dec) :-
        strip_zeros(Bool,Bool1),
        reversebool(Bool1,[],Bool2),
        build_dec(Bool2,1,0,Dec).

strip_zeros([0|R],Boo1) :-
        strip_zeros(R,Bool).
strip_zeros([1|R],[1|R]).
strip_zeros([],[]) :-
        write('ERROR: in strip_zeros, no boolean number!'),
        nl,
        abort.

reversebool([],Bool,Bool).
reversebool([X|R],Bool,Bool2) :-
        conc([X],Bool,Bool1),
        reversebool(R,Bool1,Bool2).

build_dec([],_,Dec,Dec)
build_dec([0|R],N,Dec,Dec1) :-
        N1 is N * 2,
        build_dec(R,N1,Dec,Dec1).
build_dec([1|R],N,Dec,Dec2) :-
        Dec1 is Dec + N,
        N1 is N * @,
        build_dec(R,N1,Dec1,Dec2).

expand(D) :-
        startexpand,
        setupexpand(D),
        fillexpand,
        reduceformulae,
        endexpand.

startexpand :-
        write('EXPAND: expanding formulae'),
        nl.

setupexpand(D) :-
        genallinit(D,Zeroindex,Oneindex),          % as in generateall
        buildindexh1(Zeroindex,Oneindex).
```

```
buildindexh1(L,L) :-
        asserta(h1(L,[0])).
buildindexh1(Index,Oneindex) :-
        asserta(h1(Index,[0])),
        incindex(Index,Nextindex,false),        % as in generateall
        buildindexh1(Nextindex,Oneindex).

fillexpand :-
        formula(F),
        expandformula(F),
        moreexpand(F).
fillexpand.

expandformula(F) :-
        getformulalists(F,[[]],L),
        expandlists(L).

getformulalists([[]],L,L).          % convert formulae to list of index lists
getformulalists([],L,L).
getformulalists([X|R],L,L2) :-
        isalist(X),
        getindexlist(X,[[]],I),
        doconc(I,L,L1),
        getformulalists(R,L1,L2).
getformulalists(_,_,_) :-
        write('ERROR: in getformulalists'),
        nl,
        abort.

doconc(X,[[]],X).
doconc(I,L,L1) :-
        conc(I,L,L1).

getindexlist([],L,L).               % convert formulae to list of index lists
getindexlist([X|R],L,L2) :-
        convertdash(X,Reply),
        makeindex(X,Reply,L,L1),
        getindexlist(R,L1,L2).

moreexpand(F) :-
        retract(formula(F)),
        asserta(formula(F,seen)),
        fillexpand.
moreexpand(F) :-
        write('ERROR: in moreexpand, formula = '),
        write(F),
        nl,
        abort.

convertdash(X,false) :-
        X == 0.
convertdash(X,false) :-
        X == 1.
convertdash(X,true) :-
        var(X).
convertdash(_,_) :-
        write('ERROR: in convertdash.'),
        nl,
        abort.
```

```
makeindex(X,false,L,L1) :-
        buildlist([X],L,L1).
makeindex(_,true,L,L3) :-
        buildlist([0],L,L1),
        buildlist([1],L,L2),
        conc(L1,L2,L3).

buildlist(_,[],[]).
buildlist(Index,[[]],[Index]).
buildlist(Index,[X|R],[Y|S]) :-
        conc(X,Index,Y),
        buildlist(Index,R,S).

expandlists([[]]).                      % all zero's case!
expandlists([]).
expandlists([X|L]) :-
        isalist(X),
        retract(h1(X,[0])),
        asserta(h1(X,[1])),
        expandlists(L).
expandlists(X) :-
        retract(h1(X,[0])),
        asserta(h1(X,[1])).
expandlists(_) :-
        write('ERROR: in expandlists'),
        nl,
        abort.

reduceformulae :-
        retract(formula(F,seen)),
        asserta(formula(F)),
        reduceformulae.
reduceformulae.

endexpand :-
        write('Completed expanding out the formulae'),
        nl.

isalist([]).
isalist([_|[]]).
isalist([_|_]).

incindex(Index,Nextindex,false) :-
        reverse(Index,[],Revindex),
        inczeroone(Revindex,Increvindex),
        reverse(Increvindex,[],Nextindex).
incindex(_,_,true).

reverse([X|[]],L,Rev) :-
        conc([X],L,Rev).
reverse([X|R],L,Rev) :-
        conc([X],L,Rev1),
        reverse(R,Rev1,Rev).

conc([],L,L).
conc([X|L1],L2,[X|L3]) :-
        conc(L1,L2,L3).
```

```
inczeroone([0|L],[1|L]).
inczeroone([1|R],[0|S]) :-
        inczeroone(R,S).

genallinit(D,Zeroindex,Oneindex) :-
        length(Index,D),
        minimaxindex(Index,Zeroindex,Oneindex).

minimaxindex([],[],[]).
minimaxindex([_|R],[0|S],[1|T]) :-
        minimaxindex(R,S,T).

checkhh1 :-
        starthh1,
        hh1,
        leftovers,
        restoreh,
        retractall(h1),
        endhh1.

starthh1 :-
        write('CHECK H H1. '),
        write('OK, only if no differences are reported.'),
        nl.

hh1 :-
        retract(h(A,B)),
        asserta(h(A,B,seen)),
        retract(h1(C,D)),
        asserta(h1(C,D,seen)),
        comparehh1(A,B,C,D),
        hh1.
hh1.

comparehh1(A,B,C,D) :-
        compareclasses(A,B,C,D),
        compareindexes(A,B,C,D).

compareclasses(_,B,_,D) :-
        B == D.
compareclasses(A,B,C,D) :-
        write('Compare class difference is: '),
        hform(h,A,B),
        write(', '),
        hform(h1,C,D),
        write('.'),
        nl.

hform(H,X,Y) :-
        write(H),
        write('('),
        write(X),
        write(','),
        write(Y),
        write(')').

compareindexes(A,B,C,D) :-
        comparelists(A,B,C,D),
        comparelengths(A,B,C,D),
```

```
        comparevalues(A,B,C,D).

comparelists(A,_,C,_) :-
        isalist(A),
        isalist(C).
comparelists(A,B,C,D) :-
        write('Compare indexes: list difference: '),
        hform(h,A,B),
        write(', '),
        hform(h1,C,D),
        write('.'),
        nl.

isalist([]).
isalist([_|[]]).
isalist([_|_]).

comparelengths(A,_,C,_) :-
        length(A,X),
        length(C,Y),
        X = Y.
comparelengths(A,B,C,D) :-
        write('Compare indexes: length difference: '),
        hform(h,A,B),
        write(', '),
        hform(h1,C,D),
        write('.'),
        nl.

comparevalues([],_,[],_).
comparevalues([X|R],_,[Y|S],_) :-
        X == Y,
        comparevalues(R,_,S,_).
comparevalues(A,B,C,D) :-
        write('Compare indexes: values different: '),
        hform(h,A,B),
        write(', '),
        hform(h1,C,D),
        write('.'),
        nl.

leftovers :-
        leftoverh,
        leftoverh1.

leftoverh :-
        retract(h(A,B)),
        asserta(h(A,B,seen)),
        write('Leftover difference in h: '),
        hform(h,A,B),
        write('.'),
        nl,
        leftoverh.
leftoverh.

leftoverh1 :-
        retract(h1(C,D)),
        asserta(h(C,D,seen)),
        write('Leftover difference in h1: '),
```

```
        hform(h1,C,D),
        write('.'),
        nl,
        leftoverh1.
leftoverh1.

restoreh :-
        retract(h(A,B,seen)),
        asserta(h(A,B)),
        restoreh.
restoreh.

endhh1 :-
        write('Completed check h h1.'),
        nl.

displayAvlist(Avlist) :-
        startdisplay,
        runheader,
        rundisplay(Avlist),
        nl.

startdisplay :-
        nl,
        write('Display of the input attribute/values.'),
        nl,
        nl.

runheader :-
        write('ATTRIBUTES:'),
        tab(6),
        write('VALUES:'),
        nl.

rundisplay([]).
rundisplay([X|L]) :-
        outputAvline(X),
        nl,
        rundisplay(L).

outputAvline([]) :-
        nl.
outputAvline([A|V]) :-
        outputA(A),
        outputv(V).

outputA(A) :-
        write(A),
        name(A,L),
        length(L,N),
        X is 17 - N,
        tab(X).

outputv([]).
outputv([X|L]) :-
        write(X),
        tab(2),
        outputv(L).
```

```
% 2. Display the attribute value list for the resulting equation.

displayeqAvlist(Avlist) :-
        starteqdisplay,
        runheader,
        rundisplay(Avlist),
        nl.

starteqdisplay :-
        nl,
        write('Display of the resulting equation attribute/values.'),
        nl,
        nl.

% 3. Display the attribute value list for the resulting redundancy.

displayredundAvlist(Avlist) :-
        startredunddisplay,
        runheader,
        rundisplay(Avlist),
        nl.

startredunddisplay :-
        nl,
        write('Display the resulting redundancy attribute/values.'),
        nl,
        nl.

exptgen :-
        startexptgen,
        whichexpts(E),
        runexpts(E),
        endexptgen.

startexptgen :-
        write('EXPERIMENTS GENERATOR.'),
        nl.

endexptgen :-
        write('Completed experiments generator.'),
        nl.

whichexpts(E) :-
        allexpts(Reply),

allexpts(Reply) :-
        write('Perform all experiments? '),
        write('Answer y or n.'),
        nl,
        read(X),
        checkallexpts(X,Reply).

checkallexpts(n,false).
checkallexpts(y,true).
checkallexpts(_,Reply) :-
        write('Incorrect response.'),
        nl,
        allexpts(Reply).
```

```
generateall(D) :-
        retractall(h(_,_)),
        startgenall(D),
        genallinit(D,Zeroindex,Oneindex),
        genalltests(D,Zeroindex,Oneindex),
        retractall(h(_,_)),
        endgenall(D).

startgenall(D) :-
        write('GENERATE ALL.'),
        nl,
        write('Starting generation of all tests of dimension '),
        write(D),
        nl.

genallinit(D,Zeroindex,Oneindex) :-
        length(Index,D),
        minimaxindex(Index,Zeroindex,Oneindex).

minimaxindex([],[],[]).
minimaxindex([_|R],[0|S],[1|T]) :-
        minimaxindex(R,S,T).

genalltests(D,Zeroindex,Oneindex) :-
        buildindex(Zeroindex,Oneindex),
        gencubetests(D,Zeroindex,Oneindex).

buildindex(L,L) :-
        asserta(h(L,[0])).
buildindex(Index,Oneindex) :-
        asserta(h(Index,[0])),
        incindex(Index,Nextindex,false),
        buildindex(Nextindex,Oneindex).

incindex(Index,Nextindex,false) :-
        reverse(Index,[],Revindex),
        inczeroone(Revindex,Increvindex),
        reverse(Increvindex,[],Nextindex).
incindex(_,_,true).

reverse([X|[]],L,Rev) :-
        conc([X],L,Rev).
reverse([X|R],L,Rev) :-
        conc([X],L,Rev1),
        reverse(R,Rev1,Rev).

conc([],L,L).
conc([X|L1],L2,[X|L3]) :-
        conc(L1,L2,L3).

inczeroone([0|L],[1|L]).
inczeroone([1|R],[0|S]) :-
        inczeroone(R,S).

gencubetests(D,Index,Oneindex) :-
        cubetest(D),
        incclass(Index,Reply),
        nextcubetest(D,Index,Oneindex,Reply).
```

```
incclass(Index,false) :-
        retract(h(Index,[0])),
        assertz(h(Index,[1])).
incclass(Index,Reply) :-
        retract(h(Index,[1])),
        assertz(h(Index,[0])),
        incindex(Index,Nextindex,Reply),
        incclass(Nextindex,Reply).
incclass(_,true).
incclass(_,_,_) :-
        write('ERROR: in incclass, neither h(Index,[0]) '),
        write('or h(Index,[1]) exists!'),
        nl,
        abort.

nextcubetest(_,_ ,_,true).
nextcubetest(D,Index,Oneindex,false) :-
        gencubetests(D,Index,Oneindex).

cubetest(D) :-
        write('Global!'),
        listing(h), nl, nl.
        global(D).

endgenall(D) :-
        write('Completed generating all tests of dimension '),
        write(D),
        write('.'),
        nl.

:- dynamic rc/3.

buildarray(Rlength,Clength) :-          % returns row & column lengths
        startbuildarray,
        retractall(rc(_,_,_)),
        specrclength(Rlength,row),
        specrclength(Clength,column),
        createarray(Rlength,Clength,'.'),
        endbuildarray.

startbuildarray :-
        write('GRAPH MACHINE: BUILD ARRAY.'),
        nl.

specrclength(Length,Rowcol) :-
        write('What is the required '),
        write(Rowcol),
        write(' length in the range 20 to 71 inclusive?'),
        nl,
        read(X),
        checkrclength(X,Length,Rowcol).

checkrclength(Length,Length,_) :-
        Length > 19,
        Length < 72.
checkrclength(_,Length,Rowcol) :-
        write('Incorrect response.'),
        nl,
```

```
        specrclength(Length,Rowcol).

createarray(Rlength,Clength,Value) :-
        createrow(0,Rlength,0,Clength,Value).

createrow(R,R,_,_,_).
createrow(Row,Rlength,Col,Clength,Value) :-
        createcol(Row,Col,Clength,Value),
        Row1 is Row + 1,
        createrow(Row1,Rlength,Col,Clength,Value).

createcol(_,C,C,_).
createcol(Row,Col,Clength,Value) :-
        asserta(rc(Row,Col,Value)),
        Col1 is Col + 1,
        createcol(Row,Col1,Clength,Value).

domodarray(Row,Col,Value) :-
        retract(rc(Row,Col,_)),
        asserta(rc(Row,Col,Value)).

modarray(Row,Col,Value) :-
        rc(Row,Col,Value),
        Value = [],
        domodarray(Row,Col,'x').
modarray(Row,Col,_) :-
        domodarray(Row,Col,'*').

insertarrayaxes(Rlength,Clength) :-
        retract(rc(0,0,_)),
        asserta(rc(0,0,'+')),
        Ymax is Rlength // 10,
        Xmax is Clength // 10,
        insertyaxis(Ymax),
        insertxaxis(Xmax).

printgraph(Rlength,Clength) :-
        nl,
        write('Graph of row length '),
        write(Rlength),
        write(' and column length '),
        write(Clength),
        nl,
        printrow(0,Rlength,0,Clength).

insertyaxis(Ymax) :-
        L = [10,20,30,40,50,60,70],
        modyaxis(L,Ymax).

insertxaxis(Xmax) :-
        L = [10,20,30,40,50,60,70],
        modxaxis(L,Xmax).

modyaxis([X|L],Ymax) :-
        Ymax >= X,
        retract(rc(X,0,_)),
        asserta(rc(X,0,'+')),
        modyaxis(L,Ymax).
modyaxis(_,_).
```

```
modxaxis([X|L],Xmax) :-
        Xmax >= X,
        retract(rc(0,X,_)),
        asserta(rc(X,0,'+')),
        modxaxis(L,Xmax).
modxaxis(_,_).

printrow(R,R,_,_) :-
        nl,
        nl.
printrow(Row,Rlength,Col,Clength) :-
        printcol(Row,Col,Clength),
        Row1 is Row + 1,
        printrow(Row1,Rlength,Col,Clength).

printcol(_,C,C) :-
        nl.
printcol(Row,Col,Clength) :-
        rc(Row,Col,Value),
        write(Value),
        Col1 is Col + 1,
        printcol(Row,Col1,Clength).

endbuildarray :-
        write('Completed buildarray'),
        nl.

invertclass(0,1).                    % ASSUMES valid class input
invertclass(1,0).

invertindex([],[]).
invertindex([0|R],[1|S]) :-          % ASSUMES complete, NO '_', else converts
to 1!
        invertindex(R,S).
invertindex([1|R],[0|S]) :-
        invertindex(R,S).

noisegenerator(Input,Output,Noiselevel) :-
        getnoise(Input,Output,Noiselevel).  % assumes noiselevel = [0.0 to 1.0)
noisegenerator(Bool,Bool,0).
noisegenerator(Input,Output,1) :-
        flipbool(Input,Output),
        write('Noise'),
        nl.

flipbool(0,1).
flipbool(1,0).

getnoise(Input,Output,Noiselevel) :-
        maybe(Noiselevel),                    % as in library
        flipbool(Input,Output),
        write('Noise'),
        nl.
getnoise(Bool,Bool,_).

runtime(D) :-
        startruntime,
        gettrial(D,Trial),
```

```
        useformula(Trial),
        repeattrial(D).                         % contains endruntime

startruntime :-
        write('RUNTIME MACHINE.'),
        nl.

gettrial(D,Trial) :-
        inputtrial(D,Trial),
        testtrial(D,Trial).

inputtrial(D,Trial) :-
        write('Input a trial as a 0, 1, "_" list, '),
        write('of length '),
        write(D),
        nl,
        read(Trial).

testtrial(D,Trial) :-
        isalist(Trial),
        length(Trial,D),
        testlist(D,Trial).
testtrial(D,Trial) :-
        write('Incorrect input: '),
        write('it must be a list of length '),
        write(D),
        nl,
        inputtrial(D,Trial).

useformula(Trial) :-
        tryformulas(Trial,Reply),
        givetrialans(Reply),
        replaceformula.

isalist([]).
isalist([_|[]]).
isalist([_|_]).

testlist(D,Trial) :-
        trylist(Trial,Ans),
        posttestlist(D,Trial,Ans).

trylist([],true).
trylist([X|L],Reply) :-
        var(X),
        trylist(L,Reply).
trylist([1|L],Reply) :-
        trylist(L,Reply).
trylist([0|L],Reply) :-
        trylist(L,Reply).
trylist(_,false).

posttestlist(_,_,true).
posttestlist(D,Trial,false) :-
        write('Incorrect input: '),
        write('list must be composed of 0,1,"_" only'),
        nl,
        gettrial(D,Trial).
```

```
tryformulas(Trial,Reply) :-
        formula(_),
        tryoneformula(Trial,Ans),
        moreformulas(Trial,Ans,Reply).
tryformulas(_,false).

tryoneformula(Trial,true) :-
        formula(F),
        Trial == F.
tryoneformula(_,false) :-
        retract(formula(F)),
        asserta(formula(F,seen)).

moreformulas(_,true,true).
moreformulas(Trial,false,Reply) :-
        tryformulas(Trial,Reply).

givetrialans(true) :-
        write('Answer is class = 1'),
        nl.
givetrialans(false) :-
        write('Answer is class = 0'),
        nl.

repeattrial(D) :-
        write('Another trial? Answer y or n'),
        nl,
        read(X),
        tryexit(X,D).

tryexit(y,D) :-
        gettrial(D,Trial),
        useformula(Trial),
        repeattrial(D).
tryexit(n,_) :-
        endruntime.

endruntime :-
        write('Leaving runtime machine - bye.'),
        nl,
        nl.

replaceformula :-
        retract(formula(F,seen)),
        asserta(formula(F)),
        replaceformula.
replaceformula.

specclass(Class) :-                          % specify the class
        write('Specify a class as 0 or 1'),
        nl,
        read(X),
        checkclass(X,Class).

checkclass(0,0).
checkclass(1,1).
checkclass(_,Reply) :-
        write('Incorrect response.'),
        nl,
```

```
        specclass(Reply).

specdimension(Dimension,Maxdimension) :-    % specify D given maximum D
        write('Specify the dimension between 1 and '),
        write(Maxdimension),
        write(' inclusive.'),
        nl,
        read(D),
        checkspecdim(D,Dimension,Maxdimension).

checkspecdim(D,D,Maxdimension) :-
        D > 0,
        D < Maxdimension.
checkspecdim(_,Dimension,Maxdimension) :-
        write('Incorrect response.'),
        nl,
        specdimension(Dimension,Maxdimension).

specindex(D,Index) :-                       % specify an index of dimension D
        write('Specify a boolean index list of dimension '),
        write(D),
        nl,
        read(I),
        checkindex(I,D,Index).

checkindex(L,D,L) :-
        isalist(L),
        checkalongindex(L,D,0).
checkindex(_,D,Index) :-
        write('Incorrect response.'),
        nl,
        specindex(D,Index).

checkalongindex([],D,D).
checkalongindex([X|L],D,N) :-
        isbool(X),
        M is N + 1,
        checkalongindex(L,D,M).

isbool(0).
isbool(1).

specindexclass(D,Index,Class) :-    % specify an index & class of dimension D
        specindex(D,Index),
        specclass(D,Class).

dbindexclass(Index,Class,false) :-          % return false, exists already in db
        h(Index,[Class]).
dbindexclass(Index,Class,true) :- % return true, because inserts in db
        asserta(h(Index,[Class])).

getrandbool(0) :-
        maybe(0.5).                         % from random.pl library
getrandbool(1).

randclass(Class) :-
        getrandbool(Class).

randindex(D,Index,Index2) :-
```

```
        getrandbool(Bool),
        conc([Bool],Index,Index1),
        morerandindex(D,Index1,Index2).

morerandindex(D,Index,Index) :-
        length(Index,D).
morerandindex(D,Index,Index1) :-
        randindex(D,Index,Index1).

randdim(D,Maxdim) :-    % gets random integer in range 1 to Maxdim inclusive
        Max is Maxdim + 1,
        random(1,Max,D).        % as in random.pl library

randindexclass(D,Index,Class) :-
        randindex(D,[],Index),
        randclass(Class).

% Universe prediction and test sets processor
/*
 pos cases are: 1  U = P = T                    (true,true,true)
                2        (U NOT= P) & (T = P)   (true,true,false)
                3        U NOT= P NOT= T        (false,false,false)
                4        T = U & P NOT= U       (true,false,true)
*/

univ :-
        getunivD(D),
        getUPT(U,P,T),
        setup_univ(D,U,P,T),
        setup_pred(D,U,P,T),
        setup_test(D,U,P,T),
        runupt(D,U,P,T).

/*
Purpose:        To convert a boolean list containing variables
                into a list of lists corresponding to all the
                possible instantiations of those variables.
*/

convert_varstobool(Vars,Bools) :-
        length(Vars,N),
        length(L,N),
        makezero(L,Zeros),
        makebools(Zeros,[],Bools,Vars).

makebools(L,Bools,Bools2,Vars) :-
        notallones(L),
        copy_term(L,L1),
        L1 = Vars,
        conc(L1,Bools,Bools1),
        incbool(L,L2),
        makebools(L2,Bools1,Bools2,Vars).
makebools(L,Bools,Bools1,Vars) :-
        copy_term(L,L1),
        L1 = Vars,
        conc(L1,Bools,Bools1).

incbool(L,L3) :-
        reversebool(L,[],L1),
```

```
        incbits(L1,L2),
        reversebool(L2,[],L3).

incbits([],[]).
incbits([0|R],[1|R]).
incbits([1|R],[0|S]) :-
        incbits(R,S).

notallones([0|R]).
notallones([1|R]) :-
        notallones(R).

wordtobool(D,Avlist) :-
        startwordtobool,
        getattributevalues(D,Avlist),
        endwordtobool.

startwordtobool :-
        write('WORD TO BOOLEAN.'),
        nl.

endwordtobool :-
        write('Completed wordtobool input.'),
        nl.

getattributevalues(D,Avlist) :-
        getattribvaluetype(Reply),
        processtype(Reply,D,Avlist).

getattribvaluetype(Reply) :-
        write('State the type of attribute value '),
        write('input that is required.'),
        nl,
        write('Input by file, hand or composed? '),
        write('Answer f,h or c'),
        nl,
        read(X),
        checkavtype(X,Reply).

checkavtype(f,f).
checkavtype(h,h).
checkavtype(c,c).
checkavtype(_,Reply) :-
        write('Incorrect response.'),
        nl,
        getattribvaluetype(Reply).

processtype(f,D,Avlist):-
        getfiletype(D,Avlist).
processtype(h,D,Avlist) :-
        getinputtype(D,Avlist).
processtype(c,D,Avlist) :-
        getprobdim(Probdim),
        getcomposedtype(Probdim,D,Avlist).

getprobdim(Probdim) :-
        write('Input probable dimensionality required'),
        nl,
        write('eg 0.5 => D = 1 (most probable case)'),
```

```
        nl,
        write('   0.75 => D = 3 (most probable case)'),
        nl,
        read(X),
        checkprobdim(X,Probdim).

checkprobdim(X,X) :-
        X >= 0.5,
        X < 0.01.
checkprobdim(_,Probdim) :-
        write('Probable dimension should be in the range '),
        write('0.5 to 0.01'),
        nl,
        getprobdim(Probdim).


% procedure: getinputtype, ASSUMES: NO errors on input & Av's = boolean
ONLY!

getinputtype(D,Avlist) :-
        startinputtype,
        inputAvs(0,D,[],Avlist),
        endinputtype(D,Avlist).

startinputtype :-
        write('Input the attributes for the problem'),
        nl,
        write('For each attribute, input ALL POSSIBLE values'),
        nl,
        write('Present restriction: '),
        write('attributes only have BOOLEAN values!'),
        nl,
        write('Each attribute or value must be a single word'),
        nl,
        write('beginning with a lower case letter'),
        nl,
        write('MAKE NO MISTAKES!!'),
        nl.

inputAvs(D,D2,Avlist,Avlist3) :-
        inputAs(D,D1,Avlist,Avlist1),
        inputvs(Avlist1,Avlist2),
        write('More attributes to input? '),
        write('Answer y or n'),
        nl,
        read(X),
        moreinputAvs(X,D1,D2,Avlist2,Avlist3).

inputAs(D,D1,Avlist,Avlist1) :-
        write('Input an attribute'),
        nl,
        read(X),
        conc([X],Avlist,Avlist1),
        D1 is D + 1.

inputvs([X|R],Avlist) :-
        inputallvs(X,[],V),
        conc(X,V,L),
        conc(L,R,Avlist).
```

```
inputallvs(X,L,L2) :-
        inputv(X,V),
        conc([V],L,L1),
        write('More values to input? '),
        write('Answer y or n'),
        nl,
        read(Y),
        moreinputv(X,Y,L1,L2).

moreinputAvs(n,_,_,L,L).
moreinputAvs(y,D,D1,L,L1) :-
        inputAvs(D,D1,L,L1).
moreinputAvs(_,_,_,_,_) :-
        write('ERROR: in moreinputAvs'),
        nl,
        abort.

moreinputv(_,n,L,L).
moreinputv(X,y,L,L1) :-
        inputallvs(X,L,L1).
moreinputv(_,_,_,_) :-
        write('ERROR: in moreinputv'),
        nl,
        abort.

inputv([X],V) :-
        write('Input a value for attribute '),
        write(X),
        nl,
        read(V).

endinputtype(D,Avlist) :-
        displayAvlist(Avlist),
        nl,
        write('Problem dimensionality is '),
        write(D),
        nl.

% procedure: getcomposedtype

getcomposedtype(Probdim,D,Avlist) :-
        composedAvs(Probdim,0,D,[],Avlist),
        endinputtype(D,Avlist).

composedAvs(Probdim,D,D2,Avlist,Avlist3) :-
        composeAvs(D,D1,Avlist,Avlist1),
        composevs(D1,Avlist1,Avlist2),
        morecmposeAs(Probdim,X),
        morecomposeAvs(Probdim,X,D1,D2,Avlist2,Avlist3).

composeAvs(D,D1,L,L1) :-
        D1 is D + 1,
        name(D1,D1list),
        name(attribute_,Alist),
        conc(Alist,D1list,A2list),
        name(Attrib,A2list),
        conc([Attrib],L,L1).
```

```
composevs(D,[X|R],Avlist) :-
        composeallvs(D,V),
        conc(X,V,L),
        conc(L,R,Avlist).
composeallvs(D,Onetwovalue) :-
        name(D,Dlist),
        name(value_,Vlist),
        name(_1,Onelist),
        name(_2,Twolist),
        conc(Vlist,Dlist,V1list),
        conc(V1list,Onelist,V2list),
        name(Onevalue,V2list),
        conc(V1list,Twolist,V3list),
        name(Twovalue,V3list),
        conc([Onevalue],[Twovalue],Onetwovalue).

morecomposeAs(Probdim,X) :-
        randA(Probdim,X).

randA(Probdim,0) :-
        maybe(Probdim).                    % as in library
randA(_,1).

morecomposeAvs(_,n,_,_,L,L).
morecomposeAvs(Probdim,y,D,D1,L,L1) :-
        composeAvs(Probdim,D,D1,L,L1).

% procedure: getfiletype, ASSUMES: NO errors in "dataattvalue" file!

getfiletype(D,Avlist) :-
        see(dataattvalue),
        processAvfile(0,D,[],Avlist),
        see(user).

processAvfile(D,D2,Avlist,Avlist2) :-
        read(Attrib),
        read(Value1),
        read(Value2),
        Av = [Attrib,Value1,Value2],
        D1 is D + 1,
        conc1(Av,Avlist,Avlist1),
        processAv(D1,D2,Avlist1,Avlist2).

processAv(end_of_file) :- !.
processAv(D1,D2,Avlist1,Avlist2) :-
        processAvfile.

% Procedure explore(Goal,Trace,Answer)
%
% Finds answer to given goal.
% Trace is a chain of ancestor goals and rules.
% 'explore' tends to find a positive answer to a question.
% Answer is 'false' only when all the possibilities have been investigated
% and they all resulted in 'false'.

explore(Goal, _, Goal is true was 'found as fact') :-
   fact :: Goal.

% Assume only one rule about each type of goal.
```

```
explore(Goal, Trace,
        Goal is TruthValue was derived_by Rule from Answer) :-
   Rule :: if Condition then Goal,       % Rule relevant to Goal
   explore(Condition, [Goal by Rule | Trace], Answer),
   truth(Answer, TruthValue).

explore(Goal1 and Goal2, Trace, Answer) :- !,
   explore(Goal1, Trace, Answer1),
   continue(Answer1, Goal1 and Goal2, Trace, Answer).


explore(Goal1 or Goal2, Trace, Answer) :-
   exploreyes(Goal1, Trace, Answer);     % Positive answer to Goal1
   exploreyes(Goal2, Trace, Answer).     % Positive answer to Goal2

explore(Goal1 or Goal2, Trace, Answer1 and Answer2) :- !,
   not(exploreyes(Goal1, Trace, _)),
   not(exploreyes(Goal2, Trace, _)),     % No positive answer
   explore(Goal1, Trace, Answer1),       % Answer1 must be negative
   explore(Goal2, Trace, Answer2).       % Answer2 must be negative

explore(Goal, Trace, Goal is Answer was told) :-
   useranswer(Goal, Trace, Answer).      % User-suplied answer

exploreyes(Goal, Trace, Answer) :-
   explore(Goal, Trace, Answer),
   positive(Answer).

continue(Answer1, _ and Goal2, Trace, Answer) :-
   positive(Answer1),
   explore(Goal2, Trace, Answer2),
   (positive(Answer2), Answer = Answer1 and Answer2;
    negative(Answer2), Answer = Answer2).

continue(Answer1, _ and _, _, _) :-
   negative(Answer1).

truth(_ is TruthValue was _, TruthValue) :- !.

truth(Answer1 and Answer2, TruthValue) :-
   truth(Answer1, true),
   truth(Answer2, true), !,
   TruthValue = true;
   TruthValue = false.

positive(Answer) :-
   truth(Answer, true).

negative(Answer) :-
   truth(Answer, false).


% Procedure getreply(Reply)
% Auxiliary procedure

getreply(Reply) :-
   read(Answer),
   means(Answer, Meaning), !,            % Answer means something?
```

```
   Reply = Meaning;                          % yes
   nl, write('Answer unknown, try again please'), nl,  % no
   getreply(Reply).                          % Try again

means(yes, yes).
means(y, yes).
means(no, no).
means(n, no).
means(why, why).
means(w, why).

useranswer(Goal, Trace, Answer) :-
   askable(Goal),                % Can Goal be asked of the user?
   ask(Goal, Trace, Answer).         % Ask user about Goal

ask(Goal, Trace, Answer) :-
   introduce(Goal),                % Show question to user
   getreply(Reply),                % Read user's reply
   process(Reply, Goal, Trace, Answer). % Process the reply

process(why, Goal, Trace, Answer) :-    % User is asking 'why'
   showtrace(Trace),               % Show why
   ask(Goal, Trace, Answer).         % Ask again

process(yes, Goal, Trace, Answer) :-    % User says Goal is true
   Answer = true,
   askvars(Goal);                  % Ask about variables
   ask(Goal, Trace, Answer).         % Ask for more solutions

process(no, Goal, Trace, false).       % User says Goal is false

introduce(Goal) :-
   nl, write('Is it true: '),
   write(Goal), write(?), nl.

askvars(Term) :-
   var(Term), !,                   % A variable?
   nl, write(Term), write(' = '),
   read(Term).                     % Read variable's value

askvars(Term) :-
   Term =.. [Functor | Args],         % Get arguments of a structure
   askarglist(Args).               % Ask about variables in arguments

askarglist([]).

askarglist([Term | Terms]) :-
   askvars(Term),
   askarglist(Terms).

% Procedure useranswer(Goal, Trace, Answer)
%
% Generates through backtracking, user-supplied solutions to Goal.
% Trace is a chain of ancestor goals and rules used for 'why' explanation.

:- dynamic wastold/3.
:- dynamic copy/1.
:- dynamic no_positive_answers_yet/0.
:- dynamic end_answers/1.
```

```
useranswer(Goal, Trace, Answer) :-
   askable(Goal, _),              % Can Goal be asked of the user?
   freshcopy(Goal, Copy),         % Variables in Goal renamed
   useranswer(Goal, Copy, Trace, Answer, 1).

% Do not ask again about an instantiated goal

useranswer(Goal, _, _, _, N) :-
   N > 1,                         % Repeated question?
   instantiated(Goal), !,
   fail.                          % Do not ask again

% Is Goal implied true or false for all instantiations?

useranswer(Goal, Copy, _, Answer, _) :-
   wastold(Copy, Answer, _),
   instance_of(Copy, Goal), !.    % Answer to Goal implied

% Retrieve known solutions, indexed from N on, for Goal

useranswer(Goal, _, _, true, N) :-
   wastold(Goal, true, M),
   M >= N.

% Has everything already been said about Goal?

useranswer(Goal, Copy, _, _, _) :-
   end_answers(Copy),
   instance_of(Copy, Goal), !,    % Everything was already said about Goal
   fail.

% Ask the user for (more) solutions

useranswer(Goal, _, Trace, Answer, N) :-
   askuser(Goal, Trace, Answer, N).

askuser(Goal, Trace, Answer, N) :-
   askable(Goal, ExternFormat),
   format(Goal, ExternFormat, Question, [], Variables), % Get query format
   ask(Goal, Question, Variables, Trace, Answer, N).

ask(Goal, Question, Variables, Trace, Answer, N) :-
   nl,
   (Variables = [], !,                    % Introduce query
    write('Is it true: ');
    write('Any (more) solution to: ')),
   write(Question), write('?'),
   getreply(Reply), !,                    % Reply = y/n/why
   process(Reply, Goal, Question, Variables, Trace, Answer, N).

process(why, Goal, Question, Variables, Trace, Answer, N) :-
   showtrace(Trace),              % Show why
   ask(Goal, Question, Variables, Trace, Answer, N).

process(yes, Goal, _, Variables, Trace, true, N) :-
   Next is N + 1,                 % Get new free index for 'wastold'
   (askvars(Variables),
    assertz(wastold(Goal, true, N));          % Record solution
```

```
    freshcopy(Goal, Copy),                  % Copy of Goal
    useranswer(Goal, Copy, Trace, _, Next)).  % More answers?

process(no, Goal, _, _, _, false, N) :-
    freshcopy(Goal, Copy),                  % Copy of Goal
     wastold(Copy, true, _), !,        % 'no' means: no more solutions
     assertz(end_answers(Goal)),       % Mark end of answers
     fail;
     Next is N + 1,                 % Get next free index for 'wastold'
     assertz(wastold(Goal, false, Next)).     % 'no' means: no solution

format(Var, Name, Name, Vars, [Var/Name | Vars]) :-
    var(Var), !.

format(Atom, Name, Atom, Vars, Vars) :-
    atomic(Atom), !,
    atomic(Name).

format(Goal, Form, Question, Vars0, Vars) :-
    Goal =.. [Functor | Args1],
    Form =.. [Functor | Forms],
    formatall(Args1, Forms, Args2, Vars0, Vars),
    Question =.. [Functor | Args2].

formatall([], [], [], Vars, Vars).

formatall([X | XL], [F | FL], [Q | QL], Vars0, Vars) :-
    formatall(XL, FL, QL, Vars0, Vars1),
    format(X, F, Q, Vars1, Vars).

askvars([]).

askvars([Variable/Name | Variables]):-
     nl, write(Name), write(' = '),
    read(Variable),
    askvars(Variables).

showtrace([]) :-
     nl, write('This was your question'), nl.

showtrace([Goal by Rule | Trace]) :-
     nl, write('To investigate, by '),
      write(Rule), write(','),
     write(Goal),
     showtrace(Trace).

instantiated(Term) :-
    numbervars(Term, 0, 0).          % No variables in Term

% instance_of(T1, T2) means instance of T1 is T2; that is,
% term T1 is more general than T2 or equally general as T2

instance_of(Term, Term1) :-  % Instance of Term is Term1
    freshcopy(Term1, Term2),  % Copy of Term1 with fresh set of variables
     numbervars(Term2, 0, _), !,
     Term = Term2.          % This suceeds if Term1 is instance of Term

freshcopy(Term, FreshTerm) :-  % Make a copy of Term with variables renamed
    asserta(copy(Term)),
```

```
   retract(copy(FreshTerm)), !.

lastindex(0).                % Index for 'wastold' at start

nextindex(Next) :-           % Next free index for 'wastold'
   retract(lastindex(Last)), !,
   Next is Last + 1,
   assert(lastindex(Next)).

% This procedure exists in Quintus prolog.
% However it is programmed here in case required in a different prolog

numbervars(Term, N, Nplus1) :-
   var(Term), !,             % Variable?
   Term = var/N,
   Nplus1 is N + 1.

numbervars(Term, N, M) :-
   Term =.. [Functor | Args],    % Structure or atomic
   numbervars(Args, N, M).       % Number variables in arguments

% Displaying the conclusion of a consultation and 'how' explanation

present(Answer) :-
   nl, showconclusion(Answer),
   nl, write('Would you like to see how? '),
   getreply(Reply),
   (Reply = yes, !, show(Answer);     % Show solution tree
    true).

showconclusion(Answer1 and Answer2) :- !,
   showconclusion(Answer1), write(' and '),
   showconclusion(Answer2).

showconclusion(Conclusion was _) :-
   write(Conclusion).

% 'show' displays a complete solution tree

show(Solution) :-
   nl, show(Solution, 0), !.          % Indent by 0

show(Answer1 and Answer2, H) :- !,     % Indent by H
   show(Answer1, H),
   tab(H), write(and), nl,
   show(Answer2, H).

show(Answer was Found, H) :-          % Indent by H
   tab(H), writeans(Answer),          % Show conclusion
   nl, tab(H),
   write(' was '),
   show1(Found, H).                   % Show evidence

show1(Derived from Answer, H) :- !,
   write(Derived), write(' from'),    % Sow rule name
   nl, H1 is H + 4,
   show(Answer, H1).                  % Show antecedent

show1(Found, _) :-                    % Found = 'told' or 'found as fact'
```

```
   write(Found), nl.

writeans(Goal is true) :- !,
   write(Goal).                    % Omit 'is true' on output

writeans(Answer) :-                % This is negative answer
   write(Answer).

numberargs([], N, N) :- !.

numberargs([X | L], N, M) :-
   numbervars(X, N, N1),
   numberargs(L, N1, M).

expert :-
        answeryes(Question);
        answerno(Question).
        ( miss out "getquestion",
                rest = same)
expert :-
   getquestion(Question),      % Input user's question
   (answeryes(Question);       % Try to find positive answer
    answerno(Question)).       % If no positive answer then find negative

answeryes(Question) :-         % Look for positive answers to Question
   markstatus(negative),       % No positive answer yet
   explore(Question, [], Answer),   % Trace is empty
   positive(Answer),           % Look for positive answers
   markstatus(positive),       % Positive answer found
   present(Answer), nl,
   write('More solutions? '),
   getreply(Reply),            % Read user's reply
   Reply = no.                 % Otherwise backtrack to 'explore'

answerno(Question) :-          % Look for negative answer to question
   retract(no_positive_answer_yet), !, % Has there been no positive answer?
   explore(Question, [], Answer),
   negative(Answer),
   present(Answer), nl,
   write('More negative solutions? '),
   getreply(Reply),            % Read user's reply
   Reply = no.                 % Otherwise backtrack to 'explore'

markstatus(negative) :-
   assert(no_positive_answer_yet).

markstatus(positive) :-
   retract(no_positive_answer_yet), !; true.

getquestion(Question) :-
   nl, write('Question, please'), nl,
   read(Question).

% It would be nice to allow for negated goals in left_hand sides of rules
% and therefore in questions investigated by 'explore'.
% An attempt to deal with negated goals is:
%
%  explore(not(Goal), Trace, Answer) :- !,
%     explore(Goal, Trace, Answer1),
```

```
%    invert(Answer1, Answer).          % Invert truth value
%
% invert(Goal is true was Found, (not(Goal)) is false was Found).
%
% invert(Goal is false was Found, (not(Goal)) is true was Found).
%
% This if fine if Goal is instantiated.
% If it is not then problems arise due to system assuming
% universal quantification
% rather than the required existential quantification.
% However if the question explored is instantiated
% then the problem disappears.
%
% Otherwise, proper treatment is more complicated.
% Some decistions can be as follows:
%  To explore NOT Goal, explore GOAL and now:
%     if GOAL IS FALSE THEN (NOT(GOAL)) IS TRUE;
%
%     if GOAL' is a solution of GOAL
%        AND GOAL' is as general as GOAL
%     THEN (NOT(GOAL)) is fale;
%
%     if GOAL' is a solution of GOAL and
%       GOAL' is more specific than GAOL
%       then we cannot say anything definite about GOAL.
%
% We can avoid these complications  by only allowing instantiated
% negated goals.
% This can often be achieved by proper statement of rules in the knowledge
% base. But BEWARE the knowledge acquisition engine has to know how
% to do this!!

/*******************************************************
 * Procedure: unix_filer                               *
 *                                                     *
 * Accesses Unix in order to do the following:         *
 *                                                     *
 * removes <File>.old,    copies <file>.prev to <file>.old
 * removes <File>.prev, copies <file>.pres to <file>.prev
 * removes <File>.pres, copies <file>.new to <file>.pres
 * removes <File>.new,   copies <Header> to <file>.new
 *                                                     *
 * Where File is assumed to be minus its .pl           *
 * and Header is either data_header.pl or prog_header.pl
 *                                                     *
 * If File doesn't exist in any or all forms,          *
 * the Header copy to <File>.new creates it, or a subsequent copy.
 *                                                     *
 * The Header file alone must exist.                   *
 *                                                     *
 *******************************************************/


unix_filer(Header, File) :-
                nl,
                write('Removing and copying files.'), nl,
                write('Ignore any comments from the operating system...'),
                nl, nl,
                name(Header, H),
```

```
                name(File, P),
                name('.old.pl', Q),
                name('.prev.pl', R),
                name('.pres.pl', S),
                name('.new.pl', T),

                conc(P, Q, Q1),
                conc(P, R, R1),
                conc(P, S, S1),
                conc(P, T, T1),

                filer(R1, Q1),
                filer(S1, R1),
                filer(T1, S1),
                filer(H, T1),
                nl,
                write('OK. Finished removing and copying files.'),
                nl, nl.

filer(A, B) :-
                name('rm  ', P),
                conc(P, B, P1),
                name(X, P1),
                do_unix(X),

                name('cp ', R),
                conc(R, A, X1),
                name(' ', T),
                conc(X1, T, X2),
                conc(X2, B, C),
                name(C1, C),
                do_unix(C1).

do_unix(X) :-
                (unix(system(X));
                 true).

v2 :-
        main, nl,
        continuer(Reply, 'Exit program?', 'yn'),
        (Reply = yes, !;
         v2).

main :-                                 % Topmost ie main routine
        intro_main,                     % State level now at to user
        main_choice(Choice),            % Get choice of what to do
        (Choice = exit, !;              % Exit main?
         call(Choice)).                 % Do next choice

intro_main :-
        nl, nl, write('Now entering main routine for the whole system.'), nl.

main_choice(Choice) :-                  % Get choice of what to do
        write('Chose which you require: type '), nl, tab(3),
        write('"e" to exit the system'), nl, tab(3),
        write('"r" to enter the run-time subsystem'), nl, tab(3),
        write('"b" to enter the build-time subsystem'), nl, tab(3),
        write('"i" to inspect the files'), nl,
        get_reply(Choice, main).
```

```
means(e, exit, main).                    % main
means(exit, exit, main).
means(r, run, main).
means(run, run, main).
means(b, build, main).
means(build, build, main).
means(i, inspect, main).
means(inspect, inspect, main).

means(e, exit, run).                     % run choice
means(exit, exit, run).
means(g, general, run).
means(general, general, run).
means(s, special, run).
means(special, special, run).

means(e, exit, build).                   % build choice
means(exit, exit, build).
means(b, build_it, build).

means(e, exit, inspect).                 % inspect choice
means(exit, exit, inspect).
means(i, inspect_it, inspect).


get_reply(Reply, Source) :-
        read(Answer),
        means(Answer, Meaning, Source), !,
        Reply = Meaning;
        nl, write('Answer unknown, try again please'), nl,
        get_reply(Reply, Source).

cleanup_db :-
   clean_wastold,
   clean_copy,
   clean_npay,
   clean_ea,
   clean_lastindex.

clean_wastold :-
    wastold(X, Y, Z),
   retract(wastold(X, Y, Z)),
   clean_wastold, !, fail;
   true.

clean_copy :-
   copy(X),
   retract(copy(X)),
   clean_copy, !, fail;
   true.

clean_npay :-
   no_positive_answers_yet,
   retract(no_positive_answers_yet),
   clean_npay, !, fail;
   true.

clean_ea:-
```

```
   end_answers(X),
   retract(end_answers(X)),
   clean_ea, !, fail;
   true.

clean_lastindex :-
   lastindex(X),
   X > 0,
   retract(lastindex(X)),
   clean_lastindex, !, fail;
   true.

inspect :-
        main_inspect, nl,
        continuer(Reply, 'Exit inspect files subsystem?', 'yn'),
        (Reply = yes, !;
         inspect).

main_inspect :-
        intro_inspect,                  % State level now at to user
        inspect_choice(Choice),   % Get choice of what to do
        (Choice = exit, !;              % Exit inspect?
         call(Choice)).                 % Do next choice

intro_inspect :-
        nl, nl, write('Now entering top of inspect files subsystem.'), nl.

inspect_choice(Choice) :-               % Get choice of what to do
        write('Chose which you require: type '), nl, tab(3),
        write('"e" to exit the inspect files subsystem'), nl, tab(3),
        write('"i" for inspect it'), nl,
        get_reply(Choice, inspect).

inspect_it :-
        write('This is inspect it'), nl.

build :-
        main_build, nl,
        continuer(Reply, 'Exit build-time subsystem?', 'yn'),
        (Reply = yes, !;
         build).

main_build :-
        intro_build,
        continuer(Reply, 'Input sentences, enter meanings or process
sentences?', 'imp'),
        ((Reply = input, lang);
         (Reply = meaning, meanings);
         (Reply = process, process_lang)).

intro_build :-
        nl, nl, write('Now entering top of build-time subsystem.'), nl.

ka :-
        header,
        read_sent(DS),
        tell_wait,
        addtokb(DS).
```

```
header :-
        write('Input a sentence of the form:-'), nl, tab(3),
        write('<disease name>: CLIN: <symptom name1>; <symptom name2>;
                ... '),
        write('<symptom nameN>.'), nl.

addtokb(DS) :-
        tell('kb.pl'),
        write('askable(_ has _, "Patient" has "Something").'), nl, nl,
        write('fact :: X isa disease :-'), nl, tab(4),
        write('member(X, ["acute cystitis"]).'), nl, nl,
        writerule(DS),
        told.

tell_wait :-
        nl,
        write('Please wait for a moment while I process your sentence and'),
        nl,
        write('write it out to a file.'), nl.

writerule(DS) :-
        [atom(D), :, atom(CLIN), : | S] = DS,
        write('rule1 :: if'), nl, tab(11),
        nsymptoms(S),
        write('then'), nl, tab(11),
      write('Patient has '''),
        write(D),
        write('''.').

writerule(DS) :-
        [atom(D), atom(D1), :, atom(CLIN), : | S] = DS,
        write('rule1 :: if'), nl, tab(11),
        nsymptoms(S),
        write('then'), nl, tab(11),
        write('Patient has '''),
        write(D), write(' '),
        write(D1),
        write('''.').

nsymptoms(S) :-
        [atom(X), '.'] = S,
        write('Patient has '),
        write(X), nl, tab(8).

nsymptoms(S) :-
        [atom(X), ; | Rest] = S,
        write('Patient has '),
        write(X),
        write(' and'), nl, tab(11),
        nsymptoms(Rest).

continuer(Reply, X, Y) :- write(X), nl,
                          write('Type one of: '),
                          write(Y),
                          write('.'), nl,
                          getreply(Reply, Y).


getreply(Reply, Y) :- read(Answer),
```

```
                means(Answer, Reply, Y).


getreply(Reply, Y) :-  write('Answer is unknown. '),
                       write('Try again please.'), nl,
                       write('Type one of: '),
                       write(Y),
                        write('.'), nl,
                       getreply(Reply, Y).

means(y, yes, yn).
means(n, no, yn).
means(b, build, br).
means(r, run, br).
means(i, input, imp).
means(m, meaning, imp).
means(p, process, imp).
means(u, unknown, uc).
means(c, change, uc).
means(i, input, iq).
means(q, queries, iq).

not(P) :- call(P), !, fail; true.

member(X, [X | _]) :- !.
member(X, [_ | L]) :- member(X, L).

means(y, yes, run).                    % run
means(yes, yes, run).
means(n, no, run).
means(no, no, run).
means(w, why, run).
means(why, why, run).

means(e, exit, general_choice).        % general choice
means(exit, exit, general_choice).
means(c, general_expert, general_choice).
means(cont, general_expert, general_choice).
means(continue, general_expert, general_choice).

means(e, exit, special_choice).        % special choice
means(exit, exit, special_choice).
means(c, special_expert, special_choice).
means(cont, special_expert, special_choice).
means(continue, special_expert, special_choice).

means(k, keyboard, build).             % build choice
means(i, input_file, build).
means(s, screen, build).
means(o, output_file, build).

s :-
   get0(C),
   put(C),
   put(C),
   dorest(C).

dorest(46) :- !.
```

```
autogenerate :-
        startautogen,
        getalldim(All),
        getsingledim(All,D),
        rungenerate(All,D),
        endautogen.

startautogen :-
        write('AUTO-GENERATE.'),
        nl.

getalldim(Reply) :-
        write('Test all dimensions 1-4 inclusive? '),
        write('Answer y or n.'),
        nl,
        read(X),
        checkall(X,Reply).

checkall(y,true).
checkall(n,false).
checkall(_,Reply) :-
        write('Incorrect response.'),
        nl,
        getalldim(Reply).

getsingledim(true,_).
getsingledim(false,D) :-
        write('OK, testing specific dimensions.'),
        nl,
        write('Which dimension out of 1 to 4 inclusive?'),
        nl,
        read(X),
        checkdim(X,D).

checkdim(1,1).
checkdim(2,2).
checkdim(3,3).
checkdim(4,4).
checkdim(_,Reply) :-
        write('Incorrect response.'),
        nl,
        getsingledim(false,Reply).

rungenerate(false,D) :-
        generateall(D).
rungenerate(true,_) :-
        generateall(1),
        generateall(2),
        generateall(3),
        generateall(4).

endautogen :-
        write('Completed autogenerate.'),
        nl,
        nl.


/*************************************
```

```
procedure: uniqueness

uniqueness machine: checks Index against:
   stack1(Counter,Vars_position, Index)

************************************/

uniqueness(N,Vnew,Inew) :-                    % search stack
   stack1(N,_,_),
   v_match(N,Vnew,Inew).
uniqueness(N,Vnew,Inew) :-                    % end of stack
   assertz(stack1(N,Vnew,Inew)).

v_match(N,Vnew,Inew) :-                              % test for Vmatch
   stack1(N,Vnew,Iold),
   i_match(N,Vnew,Iold,Inew).
v_match(N,Vnew,Inew) :-                              % no Vmatch in
stack
   assertz(stack1(N,Vnew,Inew)).

i_match(_,_,Iold,Inew) :-                     % got already
   Inew = Iold.
i_match(N,Vnew,_,Inew) :-                     % no Imatch, cont search
   N1 is N + 1,
   uniqueness(N1,Vnew,Inew).



/*****************************************************************
*  dynamic database:     templates of various dimensions   from files      *
*                                                                          *
*  format:               t(Position, One, Zero, D).                        *
*                                                                          *
*  example:              generate the "t" file for H7:  "gent7"            *
*                                                                          *
*****************************************************************/

% generated t file for D = 7

t(0,[1,_,_,_,_,_,_],[0,_,_,_,_,_,_],7).
t(1,[_,1,_,_,_,_,_],[_,0,_,_,_,_,_],7).
t(2,[_,_,1,_,_,_,_],[_,_,0,_,_,_,_],7).
t(3,[_,_,_,1,_,_,_],[_,_,_,0,_,_,_],7).
t(4,[_,_,_,_,1,_,_],[_,_,_,_,0,_,_],7).
t(5,[_,_,_,_,_,1,_],[_,_,_,_,_,0,_],7).
t(6,[_,_,_,_,_,_,1],[_,_,_,_,_,_,0],7).


%  Which actually means:

t(0,[1,A,B,C,D,E,F],[0,B,B,C,D,E,F],7).
t(1,[A,1,B,C,D,E,F],[A,0,B,C,D,E,F],7).
t(2,[A,B,1,C,D,E,F],[A,B,0,C,D,E,F],7).
t(3,[A,B,C,1,D,E,F],[A,B,C,0,D,E,F],7).
t(4,[A,B,C,D,1,E,F],[A,B,C,D,0,E,F],7).
t(5,[A,B,C,D,E,1,F],[A,B,C,D,E,0,F],7).
t(6,[A,B,C,D,E,F,1],[A,B,C,D,E,F,0],7).

%  Where the letters each stand for a variable, function, etc.
```

```
/*************************************************************
*  dynamic database: hypercubes of various dimensions from files          *
*                                                                         *
*  example:          even parity in H^7: parityeven7                      *
*                                                                         *
*************************************************************/

% Generating even parity for dimension 7.

h([1,0,0,0,0,0,0,0]).
h([0,0,0,0,0,0,0,1]).
h([0,0,0,0,0,0,1,0]).
h([1,0,0,0,0,0,1,1]).
h([0,0,0,0,0,1,0,0]).
h([1,0,0,0,0,1,0,1]).
h([1,0,0,0,0,1,1,0]).
h([0,0,0,0,0,1,1,1]).
h([0,0,0,0,1,0,0,0]).
h([1,0,0,0,1,0,0,1]).
h([1,0,0,0,1,0,1,0]).
h([0,0,0,0,1,0,1,1]).
h([1,0,0,0,1,1,0,0]).
h([0,0,0,0,1,1,0,1]).
h([0,0,0,0,1,1,1,0]).
h([1,0,0,0,1,1,1,1]).
h([0,0,0,1,0,0,0,0]).
h([1,0,0,1,0,0,0,1]).
h([1,0,0,1,0,0,1,0]).
h([0,0,0,1,0,0,1,1]).
h([1,0,0,1,0,1,0,0]).
h([0,0,0,1,0,1,0,1]).
h([0,0,0,1,0,1,1,0]).
h([1,0,0,1,0,1,1,1]).
h([1,0,0,1,1,0,0,0]).
h([0,0,0,1,1,0,0,1]).
h([0,0,0,1,1,0,1,0]).
h([1,0,0,1,1,0,1,1]).
h([0,0,0,1,1,1,0,0]).
h([1,0,0,1,1,1,0,1]).
h([1,0,0,1,1,1,1,0]).
h([0,0,0,1,1,1,1,1]).
h([0,0,1,0,0,0,0,0]).
h([1,0,1,0,0,0,0,1]).
h([1,0,1,0,0,0,1,0]).
h([0,0,1,0,0,0,1,1]).
h([1,0,1,0,0,1,0,0]).
h([0,0,1,0,0,1,0,1]).
h([0,0,1,0,0,1,1,0]).
h([1,0,1,0,0,1,1,1]).
h([1,0,1,0,1,0,0,0]).
h([0,0,1,0,1,0,0,1]).
h([0,0,1,0,1,0,1,0]).
h([1,0,1,0,1,0,1,1]).
h([0,0,1,0,1,1,0,0]).
h([1,0,1,0,1,1,0,1]).
h([1,0,1,0,1,1,1,0]).
```

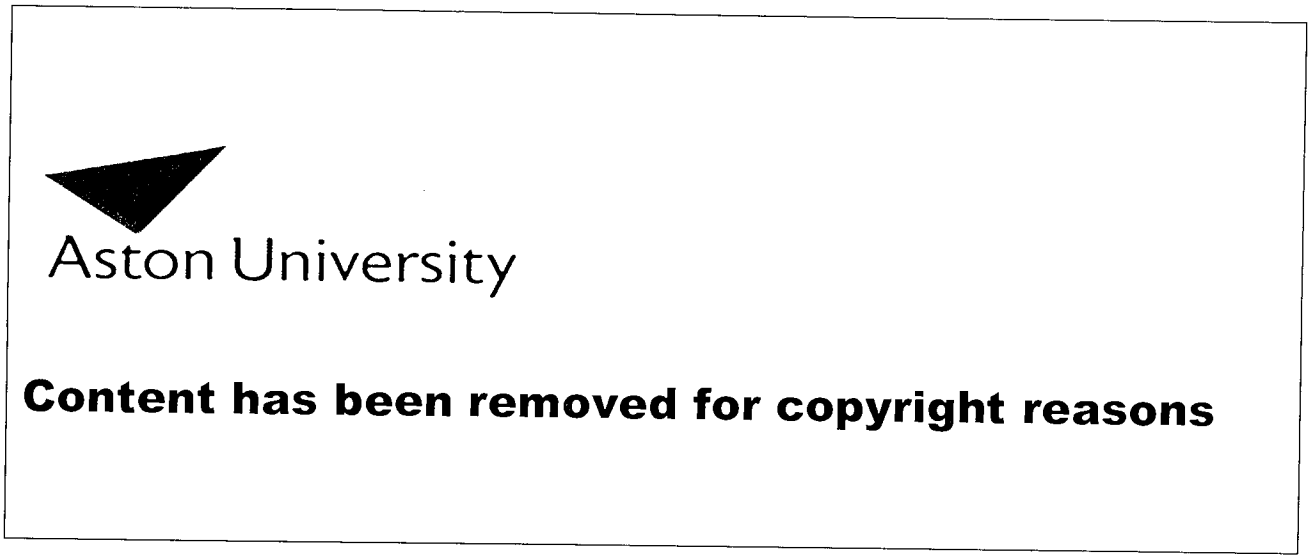
```
h([0,0,1,0,1,1,1,1]).
h([1,0,1,1,0,0,0,0]).
h([0,0,1,1,0,0,0,1]).
h([0,0,1,1,0,0,1,0]).
h([1,0,1,1,0,0,1,1]).
h([0,0,1,1,0,1,0,0]).
h([1,0,1,1,0,1,0,1]).
h([1,0,1,1,0,1,1,0]).
h([0,0,1,1,0,1,1,1]).
h([0,0,1,1,1,0,0,0]).
h([1,0,1,1,1,0,0,1]).
h([1,0,1,1,1,0,1,0]).
h([0,0,1,1,1,0,1,1]).
h([1,0,1,1,1,1,0,0]).
h([0,0,1,1,1,1,0,1]).
h([0,0,1,1,1,1,1,0]).
h([1,0,1,1,1,1,1,1]).
h([0,1,0,0,0,0,0,0]).
h([1,1,0,0,0,0,0,1]).
h([1,1,0,0,0,0,1,0]).
h([0,1,0,0,0,0,1,1]).
h([1,1,0,0,0,1,0,0]).
h([0,1,0,0,0,1,0,1]).
h([0,1,0,0,0,1,1,0]).
h([1,1,0,0,0,1,1,1]).
h([1,1,0,0,1,0,0,0]).
h([0,1,0,0,1,0,0,1]).
h([0,1,0,0,1,0,1,0]).
h([1,1,0,0,1,0,1,1]).
h([0,1,0,0,1,1,0,0]).
h([1,1,0,0,1,1,0,1]).
h([1,1,0,0,1,1,1,0]).
h([0,1,0,0,1,1,1,1]).
h([1,1,0,1,0,0,0,0]).
h([0,1,0,1,0,0,0,1]).
h([0,1,0,1,0,0,1,0]).
h([1,1,0,1,0,0,1,1]).
h([0,1,0,1,0,1,0,0]).
h([1,1,0,1,0,1,0,1]).
h([1,1,0,1,0,1,1,0]).
h([0,1,0,1,0,1,1,1]).
h([0,1,0,1,1,0,0,0]).
h([1,1,0,1,1,0,0,1]).
h([1,1,0,1,1,0,1,0]).
h([0,1,0,1,1,0,1,1]).
h([1,1,0,1,1,1,0,0]).
h([0,1,0,1,1,1,0,1]).
h([0,1,0,1,1,1,1,0]).
h([1,1,0,1,1,1,1,1]).
h([1,1,1,0,0,0,0,0]).
h([0,1,1,0,0,0,0,1]).
h([0,1,1,0,0,0,1,0]).
h([1,1,1,0,0,0,1,1]).
h([0,1,1,0,0,1,0,0]).
h([1,1,1,0,0,1,0,1]).
h([1,1,1,0,0,1,1,0]).
h([0,1,1,0,0,1,1,1]).
h([0,1,1,0,1,0,0,0]).
h([1,1,1,0,1,0,0,1]).
```

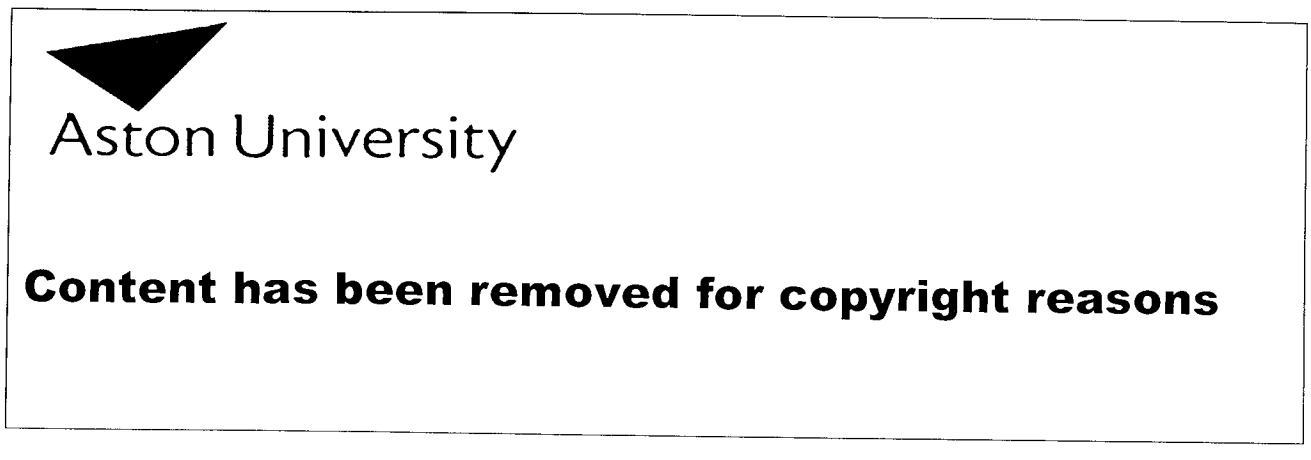
```
h([1,1,1,0,1,0,1,0]).
h([0,1,1,0,1,0,1,1]).
h([1,1,1,0,1,1,0,0]).
h([0,1,1,0,1,1,0,1]).
h([0,1,1,0,1,1,1,0]).
h([1,1,1,0,1,1,1,1]).
h([0,1,1,1,0,0,0,0]).
h([1,1,1,1,0,0,0,1]).
h([1,1,1,1,0,0,1,0]).
h([0,1,1,1,0,0,1,1]).
h([1,1,1,1,0,1,0,0]).
h([0,1,1,1,0,1,0,1]).
h([0,1,1,1,0,1,1,0]).
h([1,1,1,1,0,1,1,1]).
h([1,1,1,1,1,0,0,0]).
h([0,1,1,1,1,0,0,1]).
h([0,1,1,1,1,0,1,0]).
h([1,1,1,1,1,0,1,1]).
h([0,1,1,1,1,1,0,0]).
h([1,1,1,1,1,1,0,1]).
h([1,1,1,1,1,1,1,0]).
h([0,1,1,1,1,1,1,1]).

/*************************************************************
*  dynamic database: hypercubes of various dimensions from files  *
*                                                                 *
*  example:          General purpose parity generator             *
*                                                                 *
*************************************************************/
```

APPENDIX 8

A Trace of the 68th dimensional drugs problem.

```
Quintus Prolog Release 2.5 (Sun-4, SunOS 4.0)
Copyright (C) 1990, Quintus Computer Systems, Inc. All rights reserved.
1310 Villa Street, Mountain View, California (415) 965-7700
[consulting /user/compsci/cs_rsrch/balld/prolog.ini...]
[prolog.ini consulted 0.100 sec 1,044 bytes]
```





APPENDIX 9

Andy Lowton's parity results using back propagation: