Printing Requisition/Graphic Services

18998

1. Please complete unshaded areas on form as applicable.
2. Distribute copies as follows: White and Yellow to Graphic Services. Retain Pink Copies for your records.
3. On completion of order the Yellow copy will be returned with the printed material.
4. Please direct enquiries, quoting requisition number and account number, to extension 3451.

TITLE OR DESCRIPTION: CS-81-13

DATE REQUISITIONED: May 17
DATE REQUIRED: May 18
ACCOUNT NO.: 1 2 6 4 4 3 1 0 2

REQUISITIONER–PRINT:
PHONE: 2192
SIGNING AUTHORITY: Sue DeAngelis

MAILING INFO – NAME: DEPT.: BLDG. & ROOM NO.: ☐ DELIVER [✓ PICK-UP]

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 220
NUMBER OF COPIES: 1

TYPE OF PAPER STOCK: [✓ BOND] ☐ NCR ___ PT. ☐ COVER ☐ BRISTOL ☐ SUPPLIED ☐

PAPER SIZE: [✓ 8½ x 11] ☐ 8½ x 14 ☐ 11 x 17 ☐

PAPER COLOUR: [✓ WHITE] ☐
INK: [✓ BLACK] ☐

PRINTING: ☐ 1 SIDE ___ PGS. [✓ 2 SIDES] ___ PGS.
NUMBERING FROM TO

BINDING/FINISHING: [✓ COLLATING] [✓ STAPLING] ☐ ___ HOLE PUNCHED ☐ PLASTIC RING

FOLDING/PADDING
CUTTING SIZE

Special Instructions

COPY CENTRE — OPER. NO. / BLDG. / MACH. NO.

DESIGN & PASTE-UP — OPER. NO. / TIME / LABOUR CODE

OPER. NO.	TIME	LABOUR CODE
		D01
		D01
		D01

TYPESETTING — QUANTITY

	QUANTITY			
PAP000000				T01
PAP000000				T01
PAP000000				T01

PROOF

PRF				
PRF				
PRF				

	QUANTITY	OPER. NO.	TIME	LABOUR CODE
NEGATIVES				
FLM				C01
FLM				C01
FLM				C01
FLM				C01
FLM				C01
PMT				
PMT				C01
PMT				C01
PMT				C01
PLATES				
PLT				P01
PLT				P01
PLT				P01
STOCK				
				001
				001
				001
				001
BINDERY				
RNG				B01
RNG				B01
RNG				B01
MIS000000				B01

OUTSIDE SERVICES

$ ___ COST

TAXES – PROVINCIAL ☐ FEDERAL ☐

GRAPHIC SERV. OCT. 85 482-2

An Adaptive Plan for State-Space Problems

by

Larry Arthur Rendell

Research Report CS-81-13

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

March 1981

ABSTRACT

An adaptive plan (meta-strategy) is described which automatically generates evaluation functions for state-space problems, and which has created a function that solves the fifteen puzzle with (locally) optimal parameters. An attribute space is defined by a vector of features (functions mapping states into integers) supplied by the user. Structures (clusters) are formed in the attribute space which represent local (and often veiled) measures of performance. These regional structures constitute a refined feedback instrument for the adaptive plan.

The system is an iterative one, and each iteration consists of three steps: the solving step, the region (cluster) handling step, and the regression step. After a (one-way) graph traverser attempts a training set of problem instances (solving step), the plan clusters states in the attribute space according to probabilitistic performance statistics, via a splitting algorithm (region handling step). From the clusters, parameters for a linear evaluation function are computed (regression step). In post-initial iterations, the system's graph traverser utilizes the evaluation function generated by the preceding iteration. The region handling step refines established clusters both by revising previous probability estimates and also by further splitting, in order to improve the function progressively.

CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

1. INTRODUCTION

This thesis describes a mechanized system which adapts itself through experience to improve its performance with state-space problems. From a mathematical standpoint, there is no new theory here, rather a synthesis of mathematical tools; several concepts and methods from statistics are adapted, including probability, interval estimation (inference with known reliability), regression (least-squares fitting), and clustering (of similar data objects). The validity of the approach is evidenced by results of experiments with formula manipulation, and particularly with the fifteen puzzle.

Although the system can be understood without it, the motivation for the design is also discussed. It is possible to view the various mathematical concepts from a psychological perspective; for example clustering can be related to the goal-oriented information reduction that occurs in perception. In fact the whole approach becomes very simple when seen as a perceptual model; any complexity arises only in the mechanization of details. All paragraphs and sections of this intuitive psychological nature are marked with an asterisk, and are included for the reader who is interested in the author's conception.

An index of definitions is provided.

This introduction is organized as follows: The first section describes a well defined context around which the perceptual model can be formed: problem solving with the state-space paradigm. 1.2 narrows the domain of discourse to a particular formalization for solution search: the evaluation function, and discusses ways of optimizing the performance of such a function. Section 1.3 examines the basic structure of the system's plan for evaluation function improvement. 1.4 (*) discusses related theory of human thinking and perception. Section 1.5 introduces a necessary device for abstraction or intelligent information reduction: statistical clustering. And 1.6 begins the task of relating the adaptive system mechanism, while 1.8 considers possible alternative designs. (Briefly, section 1.7 (*) compares the system with the motivating perceptual theory.)

Let us first see what the state-space approach is and why it is used.

1.1. THE STATE-SPACE PARADIGM

Automatic problem solving is a good area in which to investigate models of human thinking and perception, because there has already been developed a theoretical structure involving state-space problems. Thus we can begin with this extant formulation and concentrate our efforts on the difficult problem of intelligence; we can conform our models to the state-space formalization and explicitly verify whether they are promising, since the experimental results can be quite conclusive.

The fifteen puzzle is a common example of a state-space problem. It is a square which contains fifteen square tiles and a space into which adjacent tiles can be slid. The goal might be:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

and the starting configuration, or starting state, any of 16!/2 (about ten trillion) even permutations of the goal state. Any state can be transformed into two, three, or four other states by means of an operator (i.e. move into the blank space the tile which is above, below, to the right, to the left). We shall return to the fifteen puzzle in depth in chapter two.[1]

State-space problems, then, are problems with explicitly describable, distinct states. From the starting state, other states can be produced by successive transformations using a specified set of operators. (Generally a state can have a number of such offspring.) The "space" in the term "state-space" refers to the entire set of states reachable from any starting state by successive applications of operators. Chapter two defines state-space problems and associated terms more precisely.

[1] When it was invented a century ago, the fifteen puzzle was extremely popular; see Ball (1931) and Gardner (1964) for additional details.

Although the fifteen puzzle is just a toy problem, it is often selected because it is easily formulated but difficult to solve by machine.[1] Many important and hard problems can be formulated as state-space problems (theorem proving for example); and progress with any representative problem is likely to be equivalent to general advancement, as long as the approach is general.

It is appropriate to represent relationships among states by using a graph. The vertices or nodes of a <u>state graph</u> are the states and a directed edge (arrow) joins one state A to another B, if B is obtained from A by an operator application. We can think of a state graph as being built by a <u>graph traverser</u>, an algorithm which begins with just the starting state vertex and <u>develops</u> nodes by applying the set of operators to them. If a goal is reached, a <u>solution</u> can be traced as a path from the goal to the starting state in the <u>final state graph</u> which results.[2]

In this process of extending the state graph (i.e. of searching through the state-space), the graph traverser is guided by some <u>search strategy</u>, which is a mechanized rule for deciding which node is to be developed next. One of the

1 Doran and Michie (1966) first used the fifteen puzzle; Doran (1967), Pohl (1969), and Chandra (1972) also worked with it. A more manageable simplification, the eight puzzle is sometimes examined -- see Schofield (1967), Michie (1967), Michie and Ross (1970), Purcell (1978), and Gaschnig (1979).

2 Our graph traverser actually constructs trees, but we retain the more general term.

simplest strategies is the exhaustive breadth-first strategy which develops nodes in the order of their generation. A breadth-first search requires prohibitive amounts of both memory and time, even for modest problems. Node growth is exponential. For example, in the fifteen puzzle, the number of states developed at level d (depth of the state graph -- i.e. path length from the starting state) is about two to the power d, so the breadth-first strategy is useless for puzzles more than eight or ten moves from the goal, whereas fifteen puzzles typically require many tens of moves. (This is a restatement of the fact that the fifteen puzzle has a very large state-space.) To be effective, a search strategy must lead the graph traverser fairly directly toward the goal, without generating too many extraneous nodes.

State-space problems and the above related concepts have been current for several years now and are well discussed in Nilsson (1971, 1980).

Early in their history, search strategies were often static and defined by the experimenter (but see Samuel, 1959). More recently, however, some interesting theoretical frameworks have been developed which facilitate dynamic modification of strategies (feedback is generally involved). Systems using such techniques can be described as learning or adaptive systems. There is a "second level" of control; the strategy modifier is called a meta-strategy or plan. Some designs are discussed in Winston (1977); see also Holland (1975).

1.2. EVALUATION FUNCTICNS AND OPTIMIZATION

One way of specifying a search strategy involves the *evaluation function*, which ranks nodes, to allow a *best-first* search. Doran & Michie (1966) originated the use of the evaluation function to guide a graph traverser. Their ideas were further explored in Doran (1968) and Michie & Ross (1970).

A simple example of an evaluation function for the fifteen puzzle is the sum, for each tile, of the distance of the tile from its "home" position, ignoring intervening tiles. (The rationale is that the goal has a zero distance score, and low scores should generally indicate proximity to the goal.) It does not work perfectly, because configurations arise with (for example) the first two rows:

2	1	3	4
5	6	7	8

Judged by the evaluation function, such a state might appear to be to be close to the goal, but in fact it is far from it. So one might use a more sophisticated function: distance score plus some constant times the number of reversals (two tiles being in correct order in a row or column except switched).

Generally for state-space problems, a function consisting of a combination of several elementary functions or *features* is beneficial. If f is a feature and A is a state, we call f(A) an *attribute* of A. Evaluation functions are always defined in terms of features (i.e. attributes are used to discriminate states).

Among other state-space problems and evaluation functions, Doran and Michie (1966) experimented with the fifteen puzzle, using the function $\theta(A) = \sum_{i=1}^{15} k_i^a p_i^b + cR$, where, in state A, k_i is the distance of the i-th tile to the empty space, p_i is the distance of the i-th tile to its proper position, and R is the number of reversals. Various values of parameter triples (a,b,c) were substituted; the best combination solved 60% of the random puzzles presented.

The ideal evaluation function (if one exists) might differentiate perfectly between states in a (short) solution, and all others. A more realistic ideal is a function which optimizes performance for all possible problem instances, or for reasonably sized random subsets.

In practice evaluation functions have been conceptualized in various ways. One aim is to estimate shortest path distance from the start to the goal, the path being constrained to include the evaluated node. This solution length scheme incorporates two components in the function f, a distance-from-starting-state component g, and a heuristic estimate h of the distance to the goal. If A is a node, f(A) = g(A) + h(A).

Hart, Nilsson and Raphael (1968) proved that if h never overestimates the remaining distance, then a graph traverser G using f is <u>admissible</u>, i.e. G will always find the shortest solution. Moreover, with an additional, mild consistency condition, G is also <u>optimal</u>, i.e. it develops a minimum number of nodes. (Unfortunately, in practice it is difficult to find an h which is both powerful and a lower bound on remaining path length.)

Pohl (1970) gereralized the form of f to be $(1-w)g + wh$ where $0 \leq w \leq 1$.[1] If $w=0$, the search is breadth-first; if $w=1$, f becomes the heuristic form alone; if $w=0.5$, f reduces to the original form of Hart et al. In his theoretical analysis, Pohl deduced several interesting results, among them that, for the worst case (least favourable evaluations by h), $w=0.5$ can do no worse than $w=1$ (in terms of number of nodes developed). He also deduced a formula which, for certain restricted cases, maps a given error bandwidth on h to an error bandwidth on number of nodes developed.

In his impressive PhD thesis, Gaschnig (1979) continued similar studies, not only with formal analyses, but also with careful interpretation of large amounts of experimental data, examining the effects of various "problem specification" and "control policy" parameters (with twc diverse problems, one of them the eight puzzle). Generalizing the theory of Pohl, Gaschnig showed that, for a certain class of heuristics, $w=0.5$ is the least costly choice for the worst case. Although he concluded that theoretical worst case

1 See also Pohl (1977).

predictions have not yet reached the stage of practical application, Gaschnig clarified some important conceptions and relationships.

A different approach to evaluation functions is to express the worth of a state as a probability measure. Slagle & Bursky (1968) thought of evaluation functions in these terms. And continuing along this line, Slagle & Farrel (1971) showed the effectiveness of automatically adjusting feature parameters using statistical feedback and regression analysis in a general graph traverser program. Also incorporating probability, Michie (1967) and Michie & Ross (1970) considered "path popularity". Our evaluation functions are defined in terms of probability. The methods of estimating the probabilities are original.

Unlike the type estimating solution length, our functions are <u>purely heuristic</u>. However, since it is possible to convert our evaluation functions into the distance estimating form, chapter seven relates some fifteen puzzle experiments to the work of Pohl and of Gaschnig.

If an evaluation function is to be composed of elementary features, they must be merged in some way. The simplest combination is linear. (So if $\underline{f} = (f_1, f_2, \ldots, f_n)$ is a <u>feature vector</u> and $\underline{b} = (b_1, b_2, \ldots, b_n)^T$ is a (column) <u>parameter vector</u>, then $\rho = \underline{f}.\underline{b}$ -- or $\log(\rho) = \underline{f}.\underline{b}$, which we use, for reasons later to be given -- provides a potentially more powerful measure (albeit a linear restriction).

Throughout this thesis, we usually assume linear relationships, but the treatment can be generalized.) Assuming that the feature vector is adequate, we now consider our original problem of creating a high performance search strategy simply as a problem of finding an optimal or close to optimal parameter vector $\underline{b}$.

If the dimensionality n is two or possibly three, $\underline{b}$ can be optimized experimentally (see Doran & Michie, 1966), but such a trial and error approach is useless for a larger number of features, particularly in view of possible nonlinearities. Because of nonlinear interactions, it is generally not feasible to optimize one parameter, subsequently add a term and set its parameter, etc., since feature interactions imply interrelationships among parameters.[1]

An alternative approach is to solve some problem instances breadth-first, measure some appropriate statistics for the resulting final state graphs, and weight the features accordingly. For any interesting problems, however, these instances would have to be the relatively easier ones, and thus not random (average prcblem instances being too difficult to solve breadth-first). But simpler problem instances may not be representative. For example it can happen that some important feature does not come into play at all, except in the case of harder instances. The second

[1] See the discussions of obstacles to adaptation in Holland (1975).

fifteen puzzle feature previously introducted, the reversal score, illustrates this phenomenon; reversals tend not to occur when a puzzle nears solution. The difficulty is a cyclic one: A good evaluation function is needed to solve hard problem instances, and information from the solution of harder instances is required in order to create a good function.

The present work is an attempt to tune parameters gradually by accumulating heuristic information over a sequence of iterations. The plan was sketched in Rendell (1977), where the main experimental results were also given. Here these results are detailed in chapter seven, where computational costs are also considered. In addition, comparisons are made with a direct, non-mechanized method of optimization.

Several other approaches had different design philosophies; some were very successful. Frequently they involved unsupervised learning, adjusting parameters during a single search. The classic example is Samuel's (1959) checker playing program, which used a linear combination of sixteen features, including relative piece advantage, control of center, etc. Michie & Ross (1970) used "pattern search" and "path popularity" in their adaptive graph traverser. Slagle & Farrel (1971) incorporated regression in their MULTIPLE (MULTIpurpose Program that Learns). And Purcell (1978) unified some earlier methods, also taking a perceptual viewpoint in a methodology to adjust a linear combination of features. In these and other cases, a potentially workable tack for evaluation function improvement has been converted into a <u>mechanized</u> plan.

Another possible approach to our optimization problem is to guess a <u>set</u> of good parameter vectors, measure the results on random problem instances, and use the extracted information to improve the parameter vector choices (according to correlations and patterns exhibited among the various vectors and performances). This process is carried out repeatedly with continually revised vector sets. Holland (1975) developed a substantial and very general theory, along with appropriate methods for such adaptive systems. His iterative design is modelled after knowledge of genetics and utilizes certain advantages of parallelism. This genetic plan benefits from the multiplicity of parameter vectors, each of whose performance is measured before a new set is generated (therein lies its main strength -- absolute optima can be located). So while the method is very effective, it seems to be inherently less efficient than a perceptual system which works with just a single parameter vector at each iteration. At the same time our system has proved useful for the two state-space problems tested.

1.3. BASIC FORM OF THE PLAN

Instead of improving the evaluation function according to its overall performance, our system examines performance statistics within local volumes, or areas, of the attribute space determined by the feature vector $\underline{f} = (f_1, f_2, \ldots, f_n)$. A "regionalization" was proposed by Michie & Ross (1970), and the present system is a realization of that intuition (c.f. also Samuel's (1967) "signature tables"). Our plan examines the ratio of the number of solution states to the total number of states which map into an attribute space area, thus providing a measure of "goodness", or the elementary penetrance, a probability that depends on the search strategy used. It is a localized measure, a refinement of the term defined by Doran & Michie (1966).

We can imagine a hypothetical situation with no time constraints which would allow us to examine the penetrance of an attribute space region for breadth-first searches of all possible problem instances. This would give us the true penetrance, which is what we endeavour to estimate, by suitable alterations of elementary ones measured for non-trivial evaluation functions and problem instances which are not too difficult. The relationships between elementary and true penetrance are of paramount importance. They are also difficult to unravel. True penetrance is discussed in chapter three, and the creation and modification of elementary penetrance comprise the major part of chapters five and six. The detrimental consequences of dispensing with true penetrance estimates, and using only elementary penetrance values, are investigated in chapter seven.

These attribute space areas, together with their true penetrance estimates, constitute the basic structures for our system. From them the plan derives a _grouped penetrance_ component of the evaluation function, which is piecewise constant. (To find the grouped penetrance of a state A, just determine into which region A maps.) And the plan deduces a _modelled penetrance_ component via weighted least-squares fitting of penetrance values to (the centers of) their regions, to provide the parameter vector $\underline{b}$ for $\log(\rho) = \underline{f}.\underline{b}$. Errors associated with the estimated penetrance values supply the weights, so that smaller errors mean larger contributions. (The modelled penetrance of A is then $\rho(A) = \exp[\underline{f}(A).\underline{b}.)$ The _full penetrance_ is a weighted average of (the logarithms of) the two components. The full evaluation function is the subject of chapter four.

There is a serious obstacle to creating a plan which attempts to measure true penetrance: If breadth-first searches can give statistics only for simple problem instances, which are generally not representative of feature participation, how can the plan obtain the necessary data? Searches that are other than breadth-first, used alone, result in unpredictably biased (elementary) penetrance values. Chapter six presents a sort of "bootstrap" method to bypass this problem; the biases are removed.

There is an additional question: How should the plan decide on the shape and extent of an attribute space region, and how many regions should be considered? Section 1.5 provides an answer, which incorporates clustering, and 1.8 asks why clusters (regions) need to be used at all.

The true penetrance, and the error-weighted evaluation function, composed of grouped and modelled penetrance, are novel to this system. The solutions to the two key questions posed above are also original, and are central.

(*) Although the clustering method can be described in a purely mathematical way, the underlying ideas originated in psychological theories. The attribute space clusters to be described in sections 1.5 and 1.6 embody a number of interrelated facts and interpretations concerning human perception. For this reason, let us now consider our second key question from this intuitive point of view.

1.4. (*) RELATED COGNITIVE THEORY

Features have been incorporated extensively in mechanized perceptual systems for many years now. Samuel (1959) was one of the first to use features (and a weighted evaluation function). Since then, numerous other researchers have utilized a similar basic approach, and the feature notion is now often an unquestioned integral part of adaptive system design.[1] But this property, attribute, or feature concept existed earlier in the theories of some psychologists. Bruner et al (1956) defined an attribute as ".... any discriminable[2] feature of an event that is susceptible of some discriminable variation from event to event."[3] They noted that "Higher organisms are highly sensitive to changes in the probability relations in their environment, and will tend to use any cue (feature) that does better than chance." As these quotations suggest, humans and other animals are capable of differentiating or <u>discriminating</u> elements of their environment according to values of features.

Discrimination can be learned. For example, pigeons have been trained to respond to a given form (shape) or colour (food being the reward for pressing the correct key). Furthermore, animals can become more discerning with accumulating experience. Experiments with human subjects have shown that ability to discriminate musical semitones increases with age, and that skill in distinguishing two tones, closely spaced in frequency, improves with

1 See Hunt (1975), Holland (1975), and Duda & Hart (1973).

2 Behavioral psychology defines the term "discrimination" in terms of observable differential response among stimuli. See Deese & Hulse (1967), chapter six.

3 Psychologists conceptualized features at least as early as 1924 -- see Bruner et al (1956), chapter two.

practice.[1] In other words, <u>differential sensitivity</u> increases. This fact is consistent with common knowledge that our perceptions become more refined as experience is gained in an area.[2] Classical music can at first seem perplexing, but later on the listener notices many subtleties; he may eventually be able to identify composer or conductor without previously having heard the particular piece or performance. A practiced chess player can immediately dismiss many board configurations as untenable. Or an experienced counseling psychologist can understand and accurately predict behavior by observing nuances in voice, posture, movements, attitudes and interests.

Another aspect of discrimination which is relevant for our system is that environmental objects or events often admit numerous possible choices of discriminating features in the mind of the perceiving organism, but there are certain general tendencies which are related to <u>selective perception</u> and <u>attention</u>.[3] Experiments have shown that the more distinctive a feature is, the more likely it is to be selected over its less discriminable counterparts.[4] (Of

1 See Werner (1957) pp. 101-103.

2 James noticed this phenomenon. See the discussion in Gibson (1969) pp. 23-25.

3 See the discussions of selective perception and attention in Gibson (1969) pp. 3,4, and especially in Bindra (1976), chapter ten.

4 See Bindra (1976) pp. 210,211.

course it often occurs that more than one feature is required to distinguish an object -- for example both colour of hair and height to recognize a friend at a distance.)

In perceptual theories, discrimination is sometimes associated with <u>categorization</u>. The category into which an object can be placed is also considered to be a function of attributes, particularly in the theory of Bruner et al (1956).[1] In that work (and in our model) discrimination and categorization are considered to be strongly related, essentially part of the same process: In formation, categories are differentiated.

Also according to Bruner et al, as time passes, perception itself becomes increasingly dependent on the (internal) categories while the (external) stimuli have a decreasing effect. Similarly Bindra (1976) states that as humans develop, ".... perception becomes more internally selective or voluntary and less externally imposed or involuntary."[2] Perception is active, not passive, and it becomes moreso.[3] To a large extent, it is a function of past experience.

There is also everyday evidence for this idea that what is perceived is a matter of internal motivation. For example, someone viewing a movie might particularly notice

1 This and other theories are also discussed in Gibson (1969).

2 Bindra (1976) p. 209.

3 See Koestler (1964), book two, chapter ten; and Arbib (1972), chapter two.

characterization and acting if he is interested in human behavior, plot if he is interested in story construction or in intellectual intricacies, visual techniques if he is a photographer, and so on. Furthermore, we generally do not look for anything at all unless it is important to us for some reason. The number of seats in a theatre and their colour, texture and microstructure are all there to be seen but usually we do not bother with them. (The mere existence of the verb "to look" -- as opposed to "to see" -- suggests activity and attention.)

Selective perception is related to the fact that the amount of data in the external world is so large; we must sceen the huge quantity of sensory input in order to have enough time to interact with our environments at all.

The information reduction seems to take place through _generalization_ and _concept attainment_.[1] For example, an object in the environment that has four legs, a seat and a back is conceptualized as a chair, an abstract construct. To consider a chair in detail could take an inordinate amount of time (e.g. the wood grain might be intricate), but simultaneously a categorization is made, the object generally becomes less interesting, and attention shifts elsewhere. It is important to note that the generalization is not made haphazardly, but rather according to the interest of the organism.

1 See Bruner et al (1956).

To account for these interrelated phenomena, some researchers have postulated a ubiquitous _hierarchical organization_ (tree structure of subcomponents), thought to underlie animal nervous systems generally, in perception, thinking, and motor control. Others have shown the advantages of hierarchical systems (natural or artificial), which include efficient and effective information processing or control, and economical organization of available resources.[1] These systems have the property of progressive information reduction and abstraction toward the root of the hierarchy ("higher" or "deeper" level) and increasing detail toward the leaves of the tree (close to the envirnoment). For example, when we walk, we are normally aware just of the act, but walking is a complex task requiring balance and coordination of muscles and groups of muscles. In the hierarchical paradigm the subtasks are learned early in life; then they become automatic, habituated, and attention can be directed to "higher", more abstract levels.

Features fit neatly into the hierarchical scheme. They, themselves are abstractions; generally less information is required to describe an object in terms of a set of

1 Simon (1962) analysed "nearly decomposable" systems that have subsystems with a low frequency of relatively weak interactions among one another, which therefore permit a hierarchical organization. Koestler (1964) wove together a number of related ideas and evidence to argue convincingly that a hierarchical model is indispensible for animal nervous systems; see also Arbib (1972). Other interesting analyses were provided by Holland (1970) and Van Emden (1970). See also the discussion and references in Jackson (1974) pp. 368-372.

attributes than to represent the object itself. Furthermore, as we have seen, features can be combined into a single evaluation function -- a "deeper" abstraction.

But to state that features and evaluation functions are abstractions is not very impressive. The interesting aspect of such a scheme is the <u>mechanism for generalization</u>; e.g. a successful process of feature creation, or an effective plan for evaluation function formation. (So the "pandemonium" design of Selfridge (1958) and the "perceptron" model of Rosenblatt (1962) are intriguing. See also Purcell (1978).[1])

This brings us to one of the central questions (concerning our system) posed at the end of section 1.3: If we imagine an attribute space, each point of which is associated with a penetrance, how should the space be divided (how should the generalization take place)? The obvious answer is: according to local consistencies and disparities in the penetrance itself. (Our system "attends to" penetrance.) A region should enclose areas of more or less constant penetrance, while other volumes should have a significantly different penetrance (attributes differentiating). The clear choice for a method of implementing this discrimination is cluster analysis (clustering can simulate categorization).

1 It should also be mentioned that Newell & Simon (1972) is a comprehensive theory of human problem solving oriented toward mechanization.

1.5. MODEL FORMATION: CLUSTERING

Clustering is a statistical method for reducing large amounts of related data to a manageable, coherent, and meaningful form. In the general problem, individual data objects are combined into clusters such that (ideally) any two objects within a cluster are similar, while any two from different clusters are dissimilar (the measure of similarity being defined according to the particular interests of the designer).[1] In our case, we are ultimately interested in differentiating states according to their likelihood of being in a solution, so our measure of similarity relates directly to penetrance.

As a measure of dissimilarity in the general clustering problem, some real-valued distance function is chosen (defined over pairs of data objects). Often there is a problem of effectively reducing an attribute vector description of the data to the singly dimensioned distance. There is no such difficulty in our case, however. We want to group areas in the attribute space which have a fairly uniform penetrance, and there is a natural way to do it. Each estimate of the true penetrance has an associated error, and our distance depends on both.

Suppose u is an estimate of the true penetrance of an area r of the attribute space. And suppose that u has an

[1] A "fuzzy" approach might be explored, but we use discrete regions.

associated error factor e. Errors are expressed as factors of the penetrance because they are sometimes very large, often several times the penetrance itself. (The large magnitudes occur because of the uncertain nature of the estimates in our system.) The likely extremes of u in r are u/e and ue. Note that, for sensibility, $e \geq 1$; also $e = 1$ implies no error. More detail is given about error factors throughout chapters three to six.

Suppose, now, that r_1 and r_2 are two attribute space volumes, with penetrance values u_1 and u_2, and error factors e_1 and e_2. Assume $u_1 \geq u_2$. Let us consider the extremes of the penetrance values to see if they overlap. The expression u_1/e_1 is the likely lower limit of the first, and u_2e_2 is the likely upper bound of the second. We define our distance (which is not a metric) to be $\mathrm{dist}(r_1,r_2) = \log[(u_1/e_1 / (u_2e_2)] = \log(u_1) - \log(u_2) - [\log(e_1) + \log(e_2)]$. (If $u_2 > u_1$, dist is defined symmetrically.)

Note that for dist to be large, both highly divergent penetrance values and small errors are necessary. The distance is greater than zero only if we can be sure, within the error bounds, that r_1 has a higher penetrance than r_2 (or vice-versa). A positive distance implies dissimilarity. This subject is treated more fully in chapter five.

The other aspect of the general clustering problem that we need to discuss is the determination of the actual cluster set (partition). This can be a combinatorial monstrosity, since the number of ways of fitting n objects into m groups is so large.[1] However, many reasonable algorithms have been developed, of a few basic types. Two classes which are relevant in our case are the joining algorithms and the splitting ones.

A joining algorithm begins with a set of clusters, each of which contains just one object. It then proceeds to agglomerate the pair of clusters exhibiting the smallest distance, and continues in this manner until some stopping criterion is met. The search for the most similar pair can be quite expensive -- and in our case there would be additional complications which arise from the (yet to be explained) facts that the only clusters that are allowed to combine are those which are adjacent in the attribute space, and that the clusters are constrained to be rectangles.

In contrast, a splitting algorithm partitions the objects into a number of clusters, then continues to subdivide the resulting clusters until some halting criterion is satisfied. Although splitting algorithms have not found much favour, because, as Hartigan states, ".... it is difficult to decide on a compelling splitting rule.",[2] there is a very natural method of splitting in our particular case. (It has already been previewed in the preceeding discussion of the distance function.)

1 See Anderberg (1973), p.3.

2 Hartigan (1975), p. 12.

With either a joining algorithm or a splitting one, there is generally the problem of realizing an effective criterion to halt the process. (I.e. how many clusters should result in the end?) The results can be misleading if the criterion is not a good one. Again, however, this does not apply to us; the standard results naturally from the penetrance statistics and their meaning (through dist), which is shown implicitly below.

In our plan, clusters are restricted to be rectangular, with boundary planes parallel to the axes, so little information is required for their specification (just two extreme corner points). The clustering algorithm is a splitting one. A parent rectangle r is specified as input, which minimally surrounds all relevant points in the attribute space. How these points are determined is outlined in the next section, as is the fact that penetrance values and their error factors can be calculated for any subrectangle cf r. The algorithm CLUSTER tentatively divides the whole rectangle into two subrectangles, in every possible way (generating every dichotomy in each attribute space dimension), and picks out the "best" of these splits. The best split is the one whose rectangular clusters have the greatest distance from each other (are the most dissimilar); i.e. the pair r_1, r_2 is selected such that $r_1 \cup r_2 = r$ and $\text{dist}(r_1, r_2) \geq \text{dist}(r', r'')$ $\forall\, r', r''$ such that $r' \cup r'' = r$. Thus, the best tentative split is the one such that the two rectangles are "most assuredly dissimilar" with regard to penetrance.

If this largest distance is less than zero, the two rectangles are recombined (since they are then similar). If, however, the distance is greater than zero, the tentative split becomes permanent, and the whole process is repeated for each of the two new clusters, in turn. The splitting continues until no further discrimination occurs.

The output from this clustering algorithm is a rectangular partition $\{r_1, r_2, \ldots r_m\}$ of the parent rectangle r. Each cluster r_j has an associated penetrance u_j and error factor e_j. We call the triple $R_j = (r_j, u_j, e_j)$ a region (a convenient structure, sets of which carry all necessary heuristic information about a problem). The number of regions output can theoretically be large, but in practice the set is quite small, often zero to three.

CLUSTER has some interesting properties. Its speed decreases reasonably slowly with the size of the feature vector. Although it is often not the case in clustering, our particular algorithm is highly "objective" -- the number, size, shape, and orientation of the individual rectangles of the partition depend largely on the statistical data. Chapter five discusses these and other aspects of the algorithm.

Our distance function is novel and clustering has not been used in adaptive problem solving systems previously.

Note that we have been discussing three separate spaces. One is the state-space. Another is the attribute space into which the states are mapped, and which is partitioned into local regions. And the third space is the one dimensional ranking space in which the evaluation function (derived from the regions) orders states.

1.6. OVERVIEW OF THE PLAN

Let us now examine the system as a whole. It is an iterative system which accumulates penetrance information. Before the first iteration begins, the user must provide a vector of integer valued features which define an attribute space to be utilized throughout as a context for clustering. At the outset the entire space is an undifferentiated lump with a constant penetrance.

Each iteration takes place in three distinct steps. The first attempts tc solve problem instances, using the full evaluation function from the previous iteration. (For the first iteration the functicn is a constant.) The second step utilizes information gathered during the solving step to create or revise clusters for a new evaluation (penetrance) function. And the third step uses these clusters to compute the modelled component of the function, via least squares fitting (regression).

1.6.1. SOLVING STEP

The first step incorporates a (one-way) graph traverser[1] and attempts to solve a training set of problem instances. Initially, this is done breadth-first (since the original evaluation function is a constant), but after the first iteration, solution attempts proceed according to an evaluation function which has been created by the previous iteration. An attempt is halted if a preselected maximum number of states is generated; but whether or not a solution is found, a final state graph results (see section 1.1). As will soon become apparent, there must be a successful solution to at least one of the input problem instances, if the iteration is to be useful.

This final state graph set determines the penetrance values, as follows: Ignoring edges, each developed node of every final state graph is mapped into a point in the attribute space.[2] Each point has a pair of integers associated with it. The total count t (for a point) is the number of developed nodes in all the final state graphs that map into that point. The other integer is the good count g, which is like the total count, except that it is further restricted to include only those nodes which appeared in a

1 Two-way graph traversers work in both directions at once; from the starting state toward the goal and from the goal to the start.

2 Conceivably, a feature might measure the structure of a state graph but those used to date have ignored generation history.

solution.

Now, the ratio g/t is the probability that a corresponding state was used in a solution of some problem instance. We call it the _elementary penetrance_. Notice that its value depends on the particular problem instance training set and also on the evaluation function. (Note, also, that if at least one problem instance is not solved, all of the good counts, hence all the elementary penetrance values, are zero.)

As an example of the solving step, for the fifteen puzzle, we might define a two-dimensional attribute space using the two features mentioned in section 1.2, the distance score and the reversal score. Then, if the search were breadth-first, points close to the origin would be found to have high g/t ratios, and points away from the origin would generally have low elementary penetrance values, since both low distance scores and few reversals are desirable.

1.6.2. REGION HANDLING STEP

This second step of an iteration uses the attribute vectors and their associated counts (of the solving step) either for clustering (first iteration), or for revising and refining previously established clusters (succeeding iterations). In either case, the definitions of the good and total counts are generalized to refer to all points lying within a particular attribute space area, rather than just to a single point. Thus the <u>elementary penetrance of a rectangle</u> (for a particular problem instance set and evaluation function) is its good count divided by its total count.[1]

In addition to the penetrance itself, it is desirable to know how reliable the value is. Recall that the error plays a part in the distance function, dist, thus in CLUSTER, and also in evaluation function formation. One source of error is a random element which relates to the magnitudes of the counts. For example, a penetrance of 0.1 might be calculated from a count ratio of 1/10 or 10/100 but the latter is more dependable. As well as this, there are other contributions to the total error, such as biases caused by the evaluation function in obtaining the counts. The various error factors are combined multiplicatively; i.e. if e_1 and e_2 are two error factors relating to a single penetrance, the combined error is e_1e_2. But the full explanation of all error factors is deferred to later chapters. For our present purposes it is enough to know that we can estimate a combined error factor for a region.

1 Since a rectangle represents an area of roughly constant penetrance, the variation of penetrance with a feature should perhaps not be too erratic in practice.

2 Error factors and regions were introduced in section 1.4.

Chapter three discusses count functions, penetrance and error factors.

1.6.2.1. FIRST ITERATION

Let us consider how the region handling step works in the first iteration. To begin, a rectangle r is formed which minimally encloses the attribute maps from the final state graphs of the solution step. This becomes the parent rectangle for CLUSTER. The penetrance used by dist is simply the elementary penetrance. CLUSTER outputs a set of regions $C_1 = \{(r_1, u_1, e_1), (r_2, u_2, e_2), \ldots\ldots, (r_m, u_m, e_m)\}$ (where u_j is the count ratio g_j/t_j for r_j). Since the search is breadth-first in the first iteration, each elementary penetrance u_j becomes an estimate of the true penetrance of r_j. (Recall the definition of true penetrance in 1.3.) So C_1 becomes the cumulative (or established) region set of iteration one. It is these cumulative regions that are the primary structures for the plan; they house the entirety of the accumulated experience (and are revised from iteration to iteration).

In our two-dimensional attribute space example for the fifteen puzzle, we typically find (what becomes) a cumulative set for the first iteration, of three regions, whose true penetrance estimates are 0.5, 0.02, and 0.001. Incidentally, we also find that all the splitting occurs in the dimension corresponding to the distance score. This is an example of the phenomenon discussed in section 1.2, that simpler problem instances are not always representative, and so cannot necessarily be used (alone) to optimize parameters.

1.6.2.2. POST-INITIAL ITERATIONS

In iterations after the first, the elementary penetrance is used again, but there is an important consequence of the fact that the evaluation function is no longer trivial. Suppose that r is some rectangle in the attribute space. We know that the elementary penetrance $u = g/t$ of r depends on the final state graph set of the solving step, and thus on the evaluation function that guides the search. Thus we can no longer assume that the elementary penetrance is a good measure of the true one (the latter defined in terms of breadth-first search). In fact it is generally a very poor estimate.

Let us examine the reasons for this assertion. Suppose that $u_a = g_a/t_a$ is an elementary penetrance of r fcr a single training problem instance, and for a non-trivial evaluation function θ; and that $u_0 = g_0/t_0$ is an elementary penetrance of r for the same problem instance, but for a breadth-first search (corresponding to the trivial function θ_0 = constant). Assume, for simplicity, that the same solution is discovered in both cases. Then $g_a = g_0$. Now, if θ is a useful evaluation function, it "wastes" fewer nodes than θ_0; i.e. if r encloses all attribute vectors, $t_a < t_0$. (It could also happen that $t_a > t_0$, if θ is worse than θ_0.) If r is a smaller area, the general expectation is still that $t_a < t_0$, especially if θ is reasonably good. Thus, we generally find that $u_a > u_0$. We define the important quantity, the <u>strategy-power</u> <u>factor</u> (for a problem set **P**, rectangle r, and evaluation function θ) to be $H(r,\theta,\mathbf{P}) = u_a/u_0$. (This is an original and central concept.) If **P** is representative, then higher values of $H(r,\theta,\mathbf{P})$ imply better performance of θ in r.

Now we can summarize the mechanism of the region handling step for post-initial iterations. There are three substeps. The first revises true penetrance estimates of the established regions; the second (possibly) refines (subdivides) each established region, incorporating CLUSTER again; and the third substep enlarges cumulative regions as necessary to engulf any uncovered outlying attribute space points. All three substeps utilize the established regions and corresponding new elementary penetrance values. (The reason that the second and third substeps take place after the penetrance revision is that both require true penetrance estimates and this ordering allows greater accuracy.) All three substeps (of necessity) take into account that the strategy-power factors are not unity, using bootstrap and smoothing designs.

First the <u>penetrance</u> <u>revision</u> substep: Suppose that two regions of the input set are $R_1 = (r_1, u_1, e_1)$ and $R_2 = (r_2, u_2, e_2)$. Suppose, also, that for the most recent solving step the counts for attribute space points lying within r_1 and r_2 are (g_1, t_1) and (g_2, t_2). Since the strategy-power factors are not unity, we cannot revise the established true penetrance estimates u_j by (say) taking their average with the elementary values g_j/t_j, but we can use the information indirectly. If, for example, $u_1 = u_2$ but $g_1/t_1 > g_2/t_2$ then u_1 should be adjusted upward and/or u_2 downward. The relationship between the established penetrance values and the elementary ones is such that the various strategy-power factors are quite disparate, but in chapter six we develop a method for converting the elementary values to true penetrance estimates by smoothing them against the corresponding established ones. The new estimated true penetrance becomes a mean (weighted according to errors) of the old value and the corrected elementary penetrance. Hence penetrance becomes a product of experience over all the iterations. (This revision has the additional effect of decreasing errors.)

The penetrance-revision portion of our plan theoretically allows the eventual creation of an evaluation function with a correctly proportioned parameter vector despite early imbalances caused by easier problem instances not being representative of relative feature importance.

Next the refinement substep: We have seen that CLUSTER is incorporated in the first iteration to (begin to) discriminate the initially undifferentiated attribute space. This algorithm is also used in subsequent iterations, but in a slightly different way; it is called once for each established region, and there is a complication because the strategy power factors are not equal to one. Suppose that a parent region is R = (r,u,e) (where r is its rectangle, u its estimated true penetrance, and e its error factor). And suppose that the algorithm splits r into m rectangles $r_1, r_2, \ldots, r_m$ whose counts are (g_1,t_1), (g_2,t_2), ..., (g_m,t_m). According to our previous discussion, $\Sigma g_j/\Sigma t_j$, the elementary penetrance of R, is likely larger than the true penetrance estimate u. Hence the elementary penetrance values g_j/t_j of the subrectangles r_j are generally overestimates of their true values (or at least the two are unlikely to be identical). If we could invent a way to convert $\Sigma g_j/\Sigma t_j$ to u, then proceed to treat the individual g_j/t_j in a similar manner, the required estimates would be produced.

One way to accomplish this is the following (which is a simplification of the approach taken in chapter six): Set $k = u / (\Sigma g_j/\Sigma t_j)$. Each new subregion has a true penetrance estimate $k(g_j/t_j)$. (Generally, $k << 1$.) Note that k is an estimate of the inverse of the power factor of the parent region. Thus the elementary penetrance values are converted to estimates of the true ones.

There are several reasons why this conversion is used. The relevant one here is that the new subregions of an old parent region now exist in the context of other old regions.

A typical example from an experiment with the fifteen puzzle is a case in which a parent rectangle r with true penetrance estimate $u = 0.001$ split into two, r_1 and r_2. The count pairs for r_1 and r_2 were $(g_1, t_1) = (13, 94)$ and $(g_2, t_2) = (8, 175)$, so $k = u / (\Sigma g_j / \Sigma t_j) = 0.001 / (21/269) = 0.013$. Hence the estimated strategy-power factor is $1/0.013 = 80$. And the true penetrance estimates were $0.013(13/94) = 0.002$, and $0.013(8/175) = 0.0006$.

As a more global example of the effect of the refinement substep, consider our two-dimensional attribute space example which had three regions at the end of the first iteration. In the second iteration, the number of regions typically might double, to about six; perhaps the region with the highest penetrance splitting into two (in the distance score dimension), the middle region not subdividing, and the region with the lowest penetrance splitting twice (once in each dimension). The reversal score begins to have an effect. Succeeding iterations result in a proportionately larger number of refinements along the reversal score dimension, for a time, then very few splits take place at all. After a few iterations, there might be a total of about fifteen cumulative regions.

The refinement substep facilitates introduction of features into the full penetrance function which have little or no effect in easy problem instances, but which are important for harder ones. In addition, the increase of the number of established regions expands the information that goes into the formation of both the grouped and modelled components of the full function.

The third, or extension substep of the region-handling step uses essentially similar methods. First rectangles are extended to cover new points. Generally (whenever the modelled evaluation function is non-trivial), the penetrance must be adjusted, since the centers of the rectangles are consequently shifted. After this extension and penetrance alteration, further splitting is also permitted in the new areas of the attribute space. The extension substep is described in chapter six, along with the other two substeps.

If $\mathbf{C}_{I-1}$ is the cumulative region set of iteration I-1, then $\mathbf{C}_I$, the cumulative set of iteration I, is the result of applying these three substeps to $\mathbf{C}_{I-1}$. This region handling -- formation in the first iteration and revision thereafter -- constitutes the essence of the plan.

1.6.3. REGRESSION STEP

The third step of an iteration computes the modelled component of the evaluation function (see section 1.3). Regression is the statistical technique of least squares fitting of a dependent (or response) variable, in which the investigator is ultimately interested, to a vector of independent ones, which are thought to have an influence. The aim is to create a mathematical model which can predict the dependent variable, given the independent vector. In our case, the former variable is the (logarithm of the) true penetrance estimate, and the latter is the attribute vector. The models are (log-) linear (although this restriction is not inherently required).

The complete unit of corresponding data for the regression is the established region, created by the second step of the iteration. If R = (r,u,e) is a cumulative region, the center points of r provide the independent attribute vector, and the observations are weighted as an (inverse-square) function of their (logarithmic) errors, which are diverse. (So more reliable data are more heavily weighted.)

There is a measure of the extent to which a given independent variable correlates with the response variable. Since the correlation computation can be mechanized, it facilitates what is called the stepwise regression procedure which begins with the trivial constant model and progressively adds the most highly correlated variable to the model, making a new best fit each time. The procedure terminates when it judges the remaining unintroduced independent variables to have an insignificant effect. Thus some of the parameters can be exactly zero. If A is a state, $\underline{f}$ is the feature vector, and $\underline{b}$ is the final parameter vector resulting after application of the stepwise procedure, then the modelled penetrance is $\rho(A) = \exp[f'(A).\underline{b}]$ (where $\underline{f}'$ is the vector function formed by augmenting $\underline{f}$ with the unity function to accommodate the constant parameter).

Since some of the parameters b_i can be identically zero, the regression step has the effect of deciding which features are generally relevant to penetrance discrimination among states. (The grouped component of the full evaluation function can still differentiate on the basis of a (so far) rejected feature, but to a much lesser degree.) At the same time, of course, this third step weights the accepted features.

As an example, consider the fifteen puzzle example we have previously discussed. In the first iteration, three cumulative regions result, and since there are only two features, the stepwise regression procedure can be applied. The result is that only the distance score has a non-zero parameter, since no discrimination occurred in the reversal score dimension. In later iterations this second feature enters the model.

There are significant reasons why the full evaluation function is created from both the grouped component and the modelled one. Over a series of iterations, the grouped component is more important initially, especially because a regression is not meaningful unless the number of regions is greater than the number of features in the model. In fact there are cases in which the modelled penetrance could never gain a foothold were it not for the inclusion of the grouped penetrance. Conversely, some of the later splits that occur can be anomalous, and the modelled penetrance restrains the grouped one in cases where it is erroneous. The two components form a synergy.

Chapter four details the regression step and also the full penetrance function.

Considering the system as a whole, it is a "bootstrap" operation which uses the evaluation function to increase efficiency of solution, and the solution statistics to improve the heuristic.

The results of using the plan with two different state-space problems are described in chapter seven.

1.7. (*) THE PLAN AS A PERCEPTUAL MODEL

Let us now take a cursory look at the plan as a model of perception, specifically with regard to the properties discussed in section 1.4. We have already seen that the clustering algorithm simulates both discrimination and categorization in a dynamic manner (discrimination is learned) with a set of features as a basis. (The "motivation" is the built-in drive toward higher penetrance.) If we consider the points mentioned in our discussion of cognitive theory, we have:

(1) Discrimination improves with practice (pp. 16-17). This corresponds to refinement of established regions in iterations following the first (pp. 35-37). Also, the system attempts to improve true penetrance estimates of the cumulative regions (pp. 34-35).

(2) The most distinctive features are selected preferentially (pp. 17-18). The plan automatically splits in the best attribute space dimension, this being a property of observed contingencies in the environment (final state graph data) (p. 22 ff.).

(3) Perception is active; it becomes increasingly dependent on the categories and less on the stimuli (pp. 18-19). The system necessarily exhibits a "drift" toward "seeing" only those states which are considered to be of greater true penetrance (pp. 13-14, 38-40). This may be related to discovery of local (as opposed to absolute) optima.

(4) Information is generalized and its quantity reduced (pp. 18-22). The clusters are concise (pp. 22-27).

(5) Hierarchical structures are involved (pp. 20-21). Fea-

tures are already abstractions (information is compacted) and they are integrated to form the desired indications of true penetrance (pp. 27-41). (Information is again reduced, meaningfully, according to predetermined guidelines.)

1.8. ALTERNATIVES

Although the present system has been successful, some of its mechanisms are not perfected, and there remain several alternate approaches that could be investigated. For example, an attribute space area need not necessarily be rectangular; the clustering algorithm might examine more than one dimension at a time; the graph traverser could be two-way; the penetrance revision could be more sophisticated; regions might be recombined; dimension reduction might be implemented. And perhaps one of the major improvements concerns the representative attribute vector (centroid) for a region (independent variable in the regression step). An outline of another design is given in chapter eight.

A more basic question is whether clustering need be used at all. (Should the regions and region handling step be eliminated?) To reiterate, the modelled evaluation function is presently obtained by a regression of penetrance on central attributes of regions. Instead, one could simply fit penetrance to individual attribute points. This works well in the first iteration when the elementary penetrance values are accurate estimates of the true ones, but in later iterations we are left with a choice between (1) fitting elementary penetrance directly (which does not work properly -- see the discussion of experiments with probit analysis in chapter seven) or (2) converting to true penetrance estimates. Selecting the latter, not only do we remain with the problem of mechanizing the conversion, but we have a further unfortunate choice: (1) carrying voluminous data from iteration to iteration, rather than the concise regions, or (2) regenerating true penetrance estimates from the model, against which the new elementary penetrance values can be compared for conversion. The latter, though somewhat artificial, is probably superior; however, our region scheme has the advantage of inherently more accurate total information (there is more accuracy still in the case of individual point retention, although supposedly the region plan keeps just enough). For example, incipient tendencies can be retained in the region set before they are exhibited in the (linear) modelled component. If, instead, a nonlinear model is incorporated to pick up this information, frequently the "real" information becomes confused with the spurious, and the resulting parameters cause very poor performance.

Experimentally, the region plan seems to be superior to other approaches. (This is disscussed in chapter seven.)

The approach in this thesis is to explore a single line of reasoning, in order to keep the presentation as simple as possible. Accordingly, clusters are rectangular restrictions, evaluation functions are exclusively linear combinations of features, and so on. Although some alternative details may later prove to be superior, the present goal is to show that the central ideas can provide a useful general framework for automatic parameter tuning.

2. PRELIMINARIES

This chapter introduces some basic artificial intelligence and system concepts used in later chapters.[1] The first section defines and discusses some fundamental formalizations, emphasizing those relevant to our system; then the organization of the system functions is cutlined. The second section serves to familiarize the reader both with these essential concepts, and with a particular state-space problem, the fifteen puzzle. The example begun there is used throughout the thesis for illustration.

2.1. STATE-SPACE PROBLEMS AND SYSTEM FORMALIZATIONS

Most of the basic approaches discussed in this section are elaborated in Nilsson (1971, 1980).

2.1.1. THE BASIC SYNTAX

In the context of artificial intelligence, it is not usual to define the term "problem" precisely. If we confine our inquiries to state-space problems, however, cur task becomes simpler. A _state_ is a condition or configuration of a problem which will admit an explicit, precisely defined

1 Throughout, some familiarity with statistical terminology such as the mean, standard deviation, skewness, probability distributions, regression, and cluster analysis is assumed, although some of these terms are briefly explained in chapter one. For details see Fraser (1958) and Snedecor & Cochran (1972).

description. For example, the state of a fifteen puzzle is its particular arrangement of tiles. (We also frequently use the term "state" to refer to the description of a state.) A state-space is the entire set of possible states of a problem.

One state can be transformed into another by means of an operator. An operator is a partial function on the set of states -- partial since, generally there may be some states to which an operator cannot apply. An operator set (for a problem) specifies all possible changes from one state into others. For example, the fifteen puzzle has a set of four operators: move into the empty square the tile which is (1) to the left, (2) to the right, (3) above, (4) below. Each of these operators is a partial function only, since the blank can be on an edge of the puzzle.

A starting state is a specially designated state of a problem, as is a goal (state). Suppose the set of possible starting states is **S**, and the set of possible goals is **F**. A state-space problem is the general problem of transforming a starting state $A_0 \in \mathbf{S}$ into a goal state $A_f \in \mathbf{F}$, successively applying operators of the specified set **O**.[1] A (state-space) problem instance is a triple $(A_0, \mathbf{O}, A_f)$. The problem instance is solved if a sequence of operators from **O** is found which accomplishes the transformation of the starting state A_0 into the goal state A_f.

[1] "Goal state" can be generalized to "goal state set", but this simpler definition will be adequate.

A _state graph_ is a directed graph whose vertices are states, one of which is the starting state.[1] If $o \in \mathbf{O}$ is an operator, then the graph has an edge (A,B) only if B = o(A). Our state graphs have the property that whenever (A,B) is an edge, so is (A,C), as long as $o' \in \mathbf{O}$ such that C = o'(A). One node B is a _descendent_ of another node A, and A is an _ancestor_ of B, if there is a path from A to B. If the length of the path is one, B is an _offspring_ of A, and A is the _parent_ of B. B can possibly have more than one parent, however our state graphs are actually trees; only the earliest discovered parent is made explicit (though we retain the term "graph"). When a problem instance is being attempted, initially the state graph has only one vertex (corresponding to the starting state A_0) but the graph can be continually extended by applying the entire set of operators to any node A; then A is said to be _developed_.[2] The _level of a vertex A_ is the path length from the starting state to A. A _solution_ is a path through a state graph from the starting state A_0 to the goal A_f.

A _graph traverser_ is an algorithm which develops nodes (i.e. extends the state graph).[3] Our traverser is designed

1 See Gill (1976) for elementary graph theory definitions.

2 Some systems have been designed which apply a subset of the operators, rather than to develop a node entirely -- see Michie (1967), Doran (1968) and Michie & Ross (1970).

3 The graph traverser is essentially the A^* algorithm of Hart et al (1968); see also Nilsson (1971).

to cease operation after a preselected maximum number of nodes M have been generated, or if the goal is found.[1] M is the <u>cutoff</u>. In either case, a <u>final state graph</u> (<u>fsg</u>) results (which is useful not only for tracing a solution, but also for deriving performance statistics). Such a graph has no more than M states.

A graph traverser is guided by some (<u>search</u>) <u>strategy</u>, which is a mechanized rule for choosing nodes for development. For example, the <u>breadth-first</u> strategy develops nodes in the order of their generation; it is complete and necessarily finds the shortest solution, but it is prohibitively inefficient.

For a given cutoff, problem instance, and search strategy, the final state graph is uniquely determined.

The term <u>performance</u> is often used in comparing efficiencies of search strategies. Various criteria are used. One is the number of nodes developed before a solution is found. Another criterion involves the calculation of the <u>effective branching factor</u> B, of a final state graph. B is given by $B(B^d-1)/(B-1) = T$, where d is the path length

[1] Our graph traverser is actually a one way traverser. Two way traversers work both ways, from the starting state to the goal, and from the goal state to the start. They often have an advantage over one way traversers. For example, if the search is breadth-first, and if the exponential growth of nodes is k^l, a two way traverser would generate just $2k^{d-1}$ states if the solution is at level d. Although future research might include investigation of our plan with a two way traverser, it is an unessential complication in the present system. But see Pohl (1969) for a comparative study of computational efficiency in uni- and bi-directional algorithms.

of the solution, and T is the total number of nodes generated. Still another performance measure, proposed by Doran & Michie (1966), is the _penetrance_, the solution length divided by the number of nodes developed. For any criterion, it is appropriate to compare strategies using a standard set of (possibly random) problem instances.

The _quality_ of a solution refers to some measure of its length, perhaps in comparison with the shortest possible solution (if known).

2.1.2. THE EVALUATION FUNCTION

The _evaluation function_ provides one way to specify a search strategy.[1] Such a function maps a state-space into the set of real numbers. We can call its range the _ranking space_, since a strategy using an evaluation function continually develops the state which has the highest current evaluation (or the lowest, depending on design). This is called _best-first_ search. As an example, the function θ_0 =

[1] Aside from evaluation functions, there are other means to form search strategies. For example Chandra (1972) wrote a program designed to solve the fifteen puzzle by correctly placing tiles in the outer gnomon first, then proceding to solve the reduced problem in the same manner. That program was the first to solve the fifteen puzzle. It differed from our approach in three ways: it had a two-way graph traverser (see previous footnote); the strategy used problem reduction (subgoals); and particularly, his strategy was specialized and created by the programmer (i.e. there was no mechanized plan). For some strategies admitting plans see Winston (1979).

constant defines the breadth-first strategy (if ties are resolved by order of generation). Our evaluation functions have their range restricted to the closed interval [0,1], because they represent probability estimators (of a state's being in the shortest solution). (So in this case largest is best.)

In this system, a third space is used which is intermediate between a state-space and the ranking space. A _feature_ is an integer valued function over a state-space.[1] Suppose that $F = \{f_1, f_2, \ldots, f_n\}$ is a set of features, and $\underline{f} = (f_1, f_2, \ldots, f_n)$ is a feature vector. If A is a state, then $f_i(A)$ is the ith _attribute of A_ ($1 \leq i \leq n$). And $\underline{f}(A)$ is the _attribute vector of A_. If **S** is a state-space, then the cartesian product $f_1(\mathbf{S}) \times f_2(\mathbf{S}) \times \ldots \times f_n(\mathbf{S})$ is the n-dimensional _attribute space of_ **S** (_defined by_ $\underline{f}$). The cardinality of the attribute space image $\underline{f}(\mathbf{S})$ is generally much smaller than that of its state-space.

Our evaluation functions have two components.[2] One is called the _grouped penetrance function_ ν; it is formed in the attribute space, using clustering and localized

1 It would be possible to define the domain of a feature to be {the set of nodes of a state graph} x {the set of state graphs}, i.e. to consider the history of a state, but our features will all be simpler.

2 A common scheme is to use a linear combination of both a distance-from-starting-state component and a distance-to-goal heuristic (discussed in section 1.2). In contrast, our two component evaluation function is _purely_ heuristic.

statistics of performance of previous evaluation functions. And the other component is called the <u>modelled penetrance function</u> ρ; its logarithm is a linear combination of features whose parameters are discovered by a least squares fit of logarithmic penetrance values to attributes. If $\underline{b} = (b_0, b_1, b_2, \ldots, b_n)^T$ is the (column) parameter vector, and $\underline{f} = (f_1, f_2, \ldots, f_n)$ is the feature vector, then the modelled penetrance of a state A is $\rho(A) = \exp[\underline{f}'(A) \cdot \underline{b}]$ where $\underline{f}' = (1, f_1, f_2, \ldots, f_n)$. (We use this primed notation similarly throughout.) Note that $\rho(A) \geq 0$. We can also consider ρ to have been normalized (by multiplication of $\underline{b}$ by a constant) so that $\max\limits_{A \in S} \rho(A) = 1$. Hence the range of both ν and ρ is the ranking space $[0,1]$. The combination of ν and ρ is the <u>full penetrance function</u> $\theta = c_\nu \nu + c_\rho \rho$ where c_ν and c_ρ are coefficients whose values vary. The coefficients are chosen so that the range of θ is also $[0,1]$. An example of a modelled function is given later in this chapter, and ν, ρ and θ are detailed in chapters three and four.

An entity which is used to control an activity or influence an algorithm is called a <u>structure</u>. For example $\underline{b}$ is a strategy structure which controls a graph traverser. An algorithm that alters a structure (generally using some sort of feedback) is an (<u>adaptive</u>) <u>plan</u> (a "second level" of control or "meta-strategy"). Our system contains a plan which makes use of the attribute space. A plan is <u>efficient</u> if it causes the structure to converge rapidly to a local optimum of performance. A plan is <u>effective</u> if it eventually generates an absolutely optimal structure.[1]

2.1.3. SYSTEM ORGANIZATION

The system operates iteratively in three steps. The function of the first (the "solving step") is to gather data from final state graphs. The second (the "region handling step") clusters the data in the attribute space. And the final step (the "regression step") uses the clusters to produce an evaluation function (with which the succeeding iteration generally solves more difficult problem instances to generate state graphs again). We can now detail the first step, then outline the other two.

Suppose that θ is a full evaluation function, and let P be a problem instance, while M is the cutoff. Then the <u>final state graph G (for θ, P and M)</u> is the final state graph created by a graph traverser using θ as its guide in attempting to solve P. To simplify notation, we use the same symbol, G, for the name of the graph traverser mapping (context will preclude ambiguity); we have G: {set of evaluation functions} x {set of problem instances} x {set of positive integers} --> {set of state graphs}. We often deal with a set of problem instances **P**. The mapping G can be extended: $\mathbf{G}(\theta,\mathbf{P},M) = \{G(\theta,P,M) \mid P \in \mathbf{P}\}$. We refer to this

[1] All terms in this paragraph are from Holland (1975).

set simply as **G** if there is no danger of ambiguity in the specification of evaluation function, problem instance set or cutoff. **G** = **G**(θ,**P**,M) is the <u>final state graph set (fsg set) (for θ, **P** and M)</u>.

We frequently use an integer subscript to refer to an iteration. Suppose that $\mathbf{P}_I$ is a problem instance set at iteration I, called the <u>training set of iteration I</u>. Let θ_{I-1} be the evaluation function created at iteration I-1, if I>1; or a constant, if I=1.[1] The graph traverser computes the final state graph set $\mathbf{G}_I = \mathbf{G}(\theta_{I-1}, \mathbf{P}_I, M)$. $\mathbf{G}_I$ is the <u>final state graph set (fsg set) of iteration I</u>. The associated computation is the <u>first step of iteration I</u> or the <u>solving step of iteration I</u>.

As we shall see later, and as the reader might suspect, no useful information is gained unless the solver has some success. So $\mathbf{P}_I$ must include at least one problem instance which is solvable according to θ_{I-1}. For example the first iteration must use a training set, one member of which is solvable breadth-first.

Later on, we shall encounter the second step, which forms or revises regions of similar penetrance in the attribute space, using statistics from the attribute space map of $\mathbf{G}_I$; and the third step, which computes an evaluation function θ_I, from these regions.

[1] This initial evaluation function could conceivably be chosen to be a non-constant function.

2.2. THE STANDARD ILLUSTRATIVE EXAMPLE

In order to clarify definitions, examples appear throughout this thesis. Every example chosen was from an actual experiment with the fifteen puzzle. So in the following, we define the fifteen puzzle precisely, detail the feature set for our standard example, and give an illustration of state graph development with an evaluation function.

2.2.1. THE FIFTEEN PUZZLE

The fifteen puzzle has states, each of which is a four by four array of integers, $\{0,1,2,\ldots,15\}$. All the integers except "0" are called <u>tiles</u>; "0" is the <u>blank</u>. Let A be a state of the puzzle, and let a_{ij} represent the integer in position i,j $(1 \le i,j \le 4)$. Suppose $0 = a_{pq}$. The operator set $\{o_1, o_2, o_3, o_4\}$, is defined as follows: $B = o_k(A)$ $(1 \le k \le 4)$ implies $b_{ij} = a_{ij}$, except for:
k=1: if $p \ne 1$, $b_{pq} = a_{p-1,q}$ & $b_{p-1,q} = 0$ (move tile down),
k=2: if $p \ne 4$, $b_{pq} = a_{p+1,q}$ & $b_{p+1,q} = 0$ (move tile up),
k=3: if $q \ne 1$, $b_{pq} = a_{p,q-1}$ & $b_{p,q-1} = 0$ (move tile right),
k=4: if $q \ne 4$, $b_{pq} = a_{p,q+1}$ & $b_{p,q+1} = 0$ (move tile left).
Applying one of these operators constitutes a <u>move</u>. Note that o_1 and o_2 are mutual inverses, as are o_3 and o_4.

The fifteen puzzle was invented in 1878. Johnson & Story (1879) showed that for any given goal state, just one half of all the puzzle instances are solvable, those whose tile arrangements form even permutions of the goal. See Gardner (1964) and Ball (1931) for additional details of the long history of this problem. Also see Schofield (1967) for an analysis of the smaller eight puzzle.

2.2.2. FIFTEEN PUZZLE FEATURES

The feature set for our standard example consists of four features. Let G be the goal state, with array elements g_{ij} $(1 \leq i,j \leq 4)$. If A is any state, with elements a_{ij}, the following define the features $\{f_i \mid 1 \leq i \leq 4\}$.

$f_1(A) = \sum_{i=1}^{4} \sum_{j=1}^{4}$ (if $g_{ij} = 0$ then 0 else $|i-i'| + |j-j'|$, where i' & j' are such that $a_{i'j'} = g_{ij}$).

f_1 is the <u>distance score</u>; it is the sum of the distances of the tiles from their "home" position.

$f_2(A) = \sum_{i=1}^{4} \sum_{j=1}^{3}$ (if $\exists k$ such that $a_{ij} = g_{ik} \neq 0$ and $a_{i,j+1} = g_{i,k-1} \neq 0$ then 1 else 0) $+ \sum_{j=1}^{4} \sum_{i=1}^{3}$ (if $\exists k$ such that $a_{ij} = g_{kj} \neq 0$ and $a_{i+1,j} = g_{k-1,j} \neq 0$ then 1 else 0).

f_2 is the <u>order wrong score</u>; each line (row or column) is examined and the number of occurrences is counted of two tiles being in their proper line and adjacent but out of order.

$f_3(A) = \sum_{i=1}^{3}$ (if $\{a_{i1}, a_{i2}, a_{i3}, a_{i4}\} = \{g_{i1}, g_{i2}, g_{i3}, g_{i4}\}$ and $\exists k$ such that $a_{ik} \neq g_{ik}$ then 1 else 0) + $\sum_{j=1}^{3}$ (if $\{a_{1j}, a_{2j}, a_{3j}, a_{4j}\} = \{g_{1j}, g_{2j}, g_{3j}, g_{4j}\}$ and $\exists k$ such that $a_{kj} \neq g_{kj}$ then 1 else 0).

f_3 is the <u>line wrong score</u>; the number of lines is counted (excluding the fourth) which have the correct tiles for the line, but with an incorrect order.

Let $\delta_{ij} = 1$ if $a_{ij} = g_{ij}$ and 0 otherwise.

$f_4(A) = \sum_{i=1}^{3}$ (if $\sum_{k=1}^{4} \delta_{ik} = 3$ then 1 else 0) + $\sum_{j=1}^{3}$ (if $\sum_{k=1}^{4} \delta_{kj} = 3$ then 1 else 0).

f_4 is the <u>blocked score</u>; the number of lines is counted which have exactly three of the four tiles correctly placed. (The incorrect fourth "blocks" entrance of the other proper one.)

Notice that these four features are not all independent of one another; for example f_2 and f_3 are related. Note also that f_2 and f_3 are generalizations of the reversal score of Doran and Michie (1966) (see section 1.2 and below).

Typically, with average problem instances, f_1 ranged from 0 to 60, f_2 from 0 to 4, f_3 from 0 to 2, f_4 from 0 to 6, so the total number of relevant points in the attribute space was about 4000.

(The remainder of this subsection can be ignored until chapter seven.) We introduce two more features that were used, but not for our standard example. They are related to some of the four above. First,

$$f_5(A) = \sum_{i=1}^{4}\sum_{j=1}^{3}(\text{if } a_{ij} = g_{i,j+1} \neq 0 \text{ and } a_{i,j+1} = g_{ij} \neq 0 \text{ then } 1 \text{ else } 0) + \sum_{j=1}^{4}\sum_{i=1}^{3}(\text{if } a_{ij} = g_{i+1,j} \neq 0 \text{ and } a_{i+1,j} = g_{ij} \neq 0 \text{ then } 1 \text{ else } 0).$$

f_5 is the <u>reversal score</u>.[1]

The final feature was motivated by Chandra's design and involves the outer gnomon only. Let us write the sequence $g_{41}, g_{31}, g_{21}, g_{11}, g_{12}, g_{13}, g_{14}$ as $c_1, c_2, ..., c_7$, and similarly a_{41}, ... as $b_1, b_2, ..., b_7$. Then

$$f_6(A) = \sum_{k=1}^{5}(\text{if } b_k = c_k \text{ and } b_{k+2} = c_{k+2} \text{ and } 0 \neq b_{k+1} \neq c_{k+1} \text{ then } 1 \text{ else } 0) + (\text{if } b_k = c_k \text{ for } k = 2, 3 \text{ and } 4, \text{ and } 0 \neq b_1 \neq c_1 \text{ then } 1 \text{ else } 0) + (\text{if } b_k = c_k \text{ for } k = 4, 5 \text{ and } 6, \text{ and } 0 \neq b_7 \neq c_7 \text{ then } 1 \text{ else } 0).$$

f_6 is the <u>gnomon-blocked score</u>.

2.2.3. ILLUSTRATION OF STATE GRAPH DEVELOPMENT

Now let us examine in some detail the development of a state graph. We chose a fifteen puzzle starting state A_0, and the modelled component of one of the evaluation functions created by the system: $\ln[\rho(A)] = \underline{f}(A).\underline{b} = 0.70 - 0.56f_1(A) - 2.18f_2(A) - 0.67f_4(A)$, where A is any state ($b_3$

[1] This reversal score is similar to that of Doran & Michie (1966); see the discussion in Pohl (1969), pp. 79,80.

was exactly zero). We make comparisons of node development based on ρ, with what it would be in the case of a breadth-first strategy. Figure 2.2 shows a state graph which is a subgraph of a final state graph whose solution length was 88, constructed with A_0 as the starting state. Arcs are labelled with operators; heavy lines show part of the solution; dotted lines indicate operators that would be applied in a breadth-first search but not with ρ guiding the graph traverser. The nodes labelled A_i $(0 \le i \le 4)$ are the ones in the solution; the ones labelled B_i $(1 \le i \le 5)$ are those which are not part of the solution but which are generated using ρ; and the states named C_i $(i=1,2)$ are never generated if ρ controls the traverser. (The operators giving a parent are not shown.) The values of $\log[\rho(A)]$ are indicated near the state A (as well as in table 2.2).

Figure 2.2 illustrates that a breadth-first search proliferates nodes exponentially. At level d, the total numbers of states generated are 1, 3, 7, 17, 41 (for d = 1, 2, 3, 4, 5); or about 2^d.

Let us examine the feature values of some of the states of figure 2.2. Table 2.2 shows that the starting state A_0 has its order wrong score $f_2(A_0) = 1$; this is because the third column has the "3" tile and "7" reversed. Also the blocked score $f_4(A_0) = 1$, since the first column is correct, except for tile "14". Both A_1 and B_1 have a distance score one higher than $f_1(A_0)$, since moving either the "10" or the

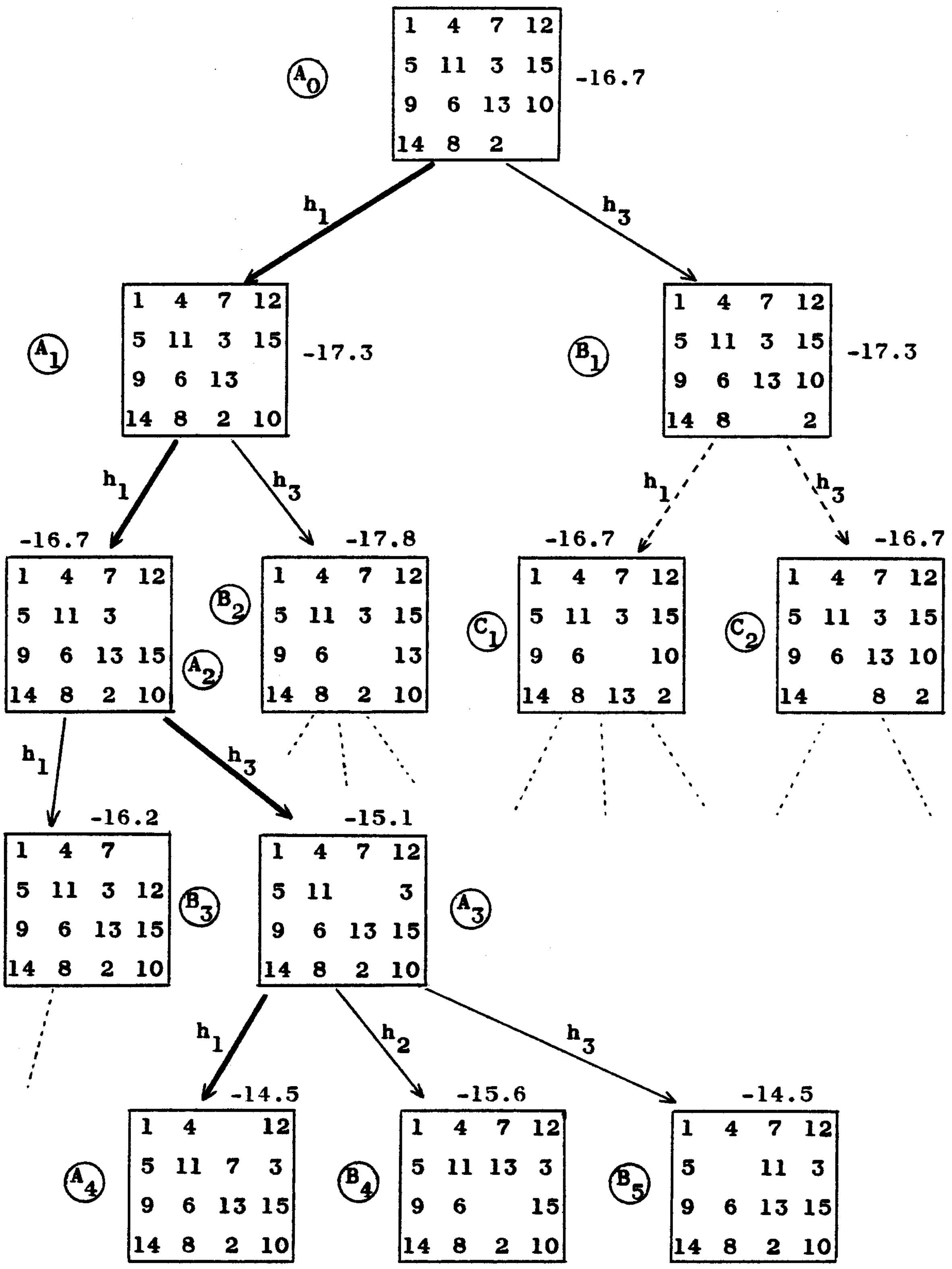


Figure 2.2. Development of a State Graph.

"2" results in a net increase of distance -- a move always increases or decreases this attribute by one. The state A_3 has $f_2(A_3) = 0$, since "3" is moved out of the third column.

Table 2.2. Attributes of States.

state A	f_1	f_2	f_3	f_4	$\ln(\rho(A))$
A_0	26	1	0	1	-16.7
A_1	27	1	0	1	-17.3
A_2	26	1	0	1	-16.7
A_3	27	0	0	1	-15.1
A_4	26	0	0	1	-14.5
B_1	27	1	0	1	-17.3
B_2	28	1	0	1	-17.8
B_3	25	1	0	1	-16.2
B_4	28	0	0	1	-15.6
B_5	26	0	0	1	-14.5
C_1	26	1	0	1	-16.7
C_2	26	1	0	1	-16.7

We can see from table 2.2 how the graph traverser selects nodes for development. At the outset, A_0 is the only state, and just two operators can be applied, resulting in A_1 and B_1. Applying ρ to A_1 and B_1 results in a tie, which is resolved according to the earliest generated node. After A_1 is developed, the ranking space contains images of A_2, B_2 and B_1 (these are the only undeveloped states). $\rho(A_2)$ is the highest. After A_2 is developed, only A_3, B_1, B_2 and B_3 are candidates. $f_1(B_3) = 25$ and $f_1(A_3) = 27$ but

$f_2(B_3) = 1$ while $f_2(A_3) = 0$ and the difference in the f_2 term ($b_2 = -2.18$) outweighs that of f_1 ($b_1 = -0.56$) in this case, so A_3 is developed next.

Although this example does not illustrate the fact, it often occurs that some state away from the eventual solution has a better evaluation, so the traverser temporarily works elsewhere.[1]

Generally, for our standard example, the training problem sets $\mathbf{P_I}$ had a dozen or fewer members; the associated final state graphs $\mathbf{G_I}$ had a couple of thousand nodes, but the attribute space maps of $\mathbf{G_I}$ never included more than a few hundred different points.

The next chapter begins to supply details concerning the penetrance, its error, and the evaluation function.

1 Gaschnig (1979) studied this "hopping around".

3. CLUSTERS, PENETRANCE AND EVALUATION FUNCTIONS

Ultimately to gain a measure of their "goodness", we would like to map states into points in an attribute space, and there form clusters. The cluster boundaries are chosen so that all points within a cluster correspond to states which have a roughly similar probability of being in a solution. Thus a single probability (penetrance) is assigned to a cluster, for use in the grouped component of the evaluation function and indirectly for the modelled component.

But first we must define the penetrance, examine its distribution, and begin to consider how to estimate "true" penetrance values and how to determine the reliability of these approximations -- or rather the opposite -- the error. In later chapters we use both the penetrance values and their errors in the actual formation of clusters and also in weighted least squares fitting for the evaluation function; but in this chapter we concentrate on the clusters themselves. In addition, we introduce the form of our modelled evaluation function and relate it to penetrance.

3.1. GROUNDWORK

We first describe a rectangular restriction of clusters, and then introduce two functions used to form and to characterize clusters.

3.1.1. RECTANGLES

In order to keep the information required for their specification to a minimum, we can restrict clusters to be rectangular. Suppose that we have an n-dimensional integer (attribute) space, **A**. Consider a rectangle r in **A** which is aligned with the axes. r may be completely defined by just two (extreme) corner points, $\underline{lp}(r) = (a_1, a_2, \ldots, a_n)$ and $\underline{up}(r) = (b_1, b_2, \ldots, b_n)$, where $\underline{lp}(r)$ is the lower, and $\underline{up}(r)$ the upper, so $a_i \leq b_i$, $i \leq n$. A point $\underline{x} = (x_1, x_2, \ldots, x_n)$ is contained in r iff $a_i \leq x_i \leq b_i$, $1 \leq i \leq n$. We can abbreviate this: $\underline{x} \in r$.

If $\underline{lp}(r) = \underline{up}(r)$, we say that r is degenerate, or r is a point rectangle.

If a finite portion of **A** is partitioned into m rectangles $\{r_1, r_2, \ldots, r_m\}$, then the whole set can be entirely specified by just 2mn integers, via m corner point pairs, $(\underline{lp}(r_j), \underline{up}(r_j))$ $(1 \leq j \leq m)$.

3.1.2. COUNT FUNCTIONS

We introduce an important pair of "count" functions over rectangles in an attribute space. These functicns have a dual purpose; they are used to construct and to modify an emerging evaluation function, and they participate in the process of clustering. Both of these roles will gradually become clearer.

Suppose we have the following:

(1) A state-space representation of a problem, and a set of k final state graphs $\mathbf{G} = \{G_1, G_2, \ldots, G_k\} = \mathbf{G}(\theta, \mathbf{P}, M)$ (with at least one success), where θ is an associated evaluation function, $\mathbf{P}$ is an associated set of k problem instances, and M is a cutoff.

(2) A feature vector $\underline{f} = (f_1, f_2, \ldots f_n)$ defining an attribute space $\mathbf{A}$.

(3) A rectangle r in $\mathbf{A}$.

We define the two integer valued <u>count functions</u>, g and t: The <u>total</u> (<u>state</u>) <u>count of r</u> (<u>for f and</u> **G**),

$t(\underline{f}, \mathbf{G}, r) \underset{def}{=\!=}$ the total number of developed nodes A, such that, for some $G \in \mathbf{G}$, A is in the graph G, and $\underline{f}(a) \in r$ (i.e. the attribute space map of A falls inside the rectangle r).

And the <u>good</u> (<u>state</u>) <u>count of r</u> (<u>for f and</u> **G**),

$g(\underline{f}, \mathbf{G}, r) \underset{def}{=\!=}$ the number of developed nodes A, such that, for some $G \in \mathbf{G}$:

(1) $\underline{f}(A) \in r$, and

(2) A is in a solution in G.

We abbreviate the counts to $g(\mathbf{G}, r)$ and $t(\mathbf{G}, r)$ if there is no danger of ambiguity in the specification of $\underline{f}$.

3.2. PENETRANCE

From the count functions we define the basis of (both components of) the evaluation function and of the clustering process.

3.2.1. PROBABILITY MEASURE

Let $\underline{f}$ be a feature vector, $\mathbf{G}$ a set of final state graphs, and r a rectangle. Consider a function mapping $\{\underline{f}\} \times \{\mathbf{G}\} \times \{r\}$ into the ranking space $[0,1]$, called the penetrance of r (for f and **G**)

$$U(\underline{f},\mathbf{G},r) \underset{def}{=\!=} g(\underline{f},\mathbf{G},r) \;/\; t(\underline{f},\mathbf{G},r).$$

For any $\mathbf{G}$, the penetrance $U(\mathbf{G},r)$ is the conditional probability that a state $A \in G \in \mathbf{G}$ is in a solution in $\mathbf{G}$, given that $\underline{f}(A) \in r$. Our penetrance is a localized refinement of the term defined by Doran & Michie (1966).

Let $\theta_0(A)$ = constant for all A, and let $\mathbf{P}_a$ be the entire set of all solvable problem instances. (If $\mathbf{P}_a$ is infinite, limit it in some reasonable way which does not disturb randomness unduly -- such as taking a large sample in place of $\mathbf{P}_a$ -- we shall assume this theoretical problem of infinite quantities has been circumvented throughout the remainder of this thesis. Similarly we avoid the academic question of whether it is feasible to determine the solvability of a problem instance.) Now define the

<u>exhaustive</u> <u>fsg</u> <u>set</u> $\mathbf{G}_a \overset{=}{\underset{def}{}} G(\theta_0, P_a, \infty)$ (i.e. the graph traverser is allowed to continue indefinitely, breadth-first, until a solution is found, this being repeated for each and every problem instance). $\mathbf{G}_a$ is the set of all shortest solution final state graphs (excluding ties because an fsg has a single solution only). The penetrance $U(\underline{f}, \mathbf{G}_a, r)$ is the <u>true</u> <u>penetrance</u> (<u>of</u> <u>r</u>).

If it could be known for all degenerate r, the true penetrance would determine an ideal evaluation function, since it is precisely the ratio of solution states to total states for all shortest solutions (except for ties).

Since $\mathbf{G}_a$ is generally extremely large, it would be reasonable to select a random subset and estimate $U(\mathbf{G}_a, r)$. For typical problems however, the calculation of a breadth-first final state graph of even a single random instance is impractical. Thus we might consider selecting easy (short solution) problem instances, rather than random ones. This could result in useful information for some problems, but generally, easy instances may not be representative (our standard example will illustrate this -- three of the four features come into play only for harder instances). This system <u>begins</u> with easy problem instances, though; then it proceeds to compute and to use repeated estimates of $U(\mathbf{G}_a, r)$ in more sophisticated ways, continually attempting to improve its accuracy.

Let us now define a working penetrance for general fsg sets **G**. In typical sample spaces, the counts are often small; particularly, the good count is frequently zero. When it is, we substitute the user-specified parameter _zval_ for g(**G**,r), whose value is intended to be close to zero. Throughout all experiments a value of 0.5 was used; this is equivalent to guessing that if the sample space were twice as large, a state A, such that $\underline{f}(A) \in r$ and A is in a solution, would actually be encountered.[1]

The _elementary penetrance of a rectangle r (for a set **G** of final state graphs and feature vector f)_

$$W(\underline{f},\mathbf{G},r) \underset{\text{def}}{=\!=} \begin{cases} g(\underline{f},\mathbf{G},r) \;/\; t(\underline{f},\mathbf{G},r) \;, & \text{if } g(\underline{f},\mathbf{G},r)>0 \\ zval \;/\; t(\underline{f},\mathbf{G},r) \;, & \text{if } g(\underline{f},\mathbf{G},r)=0. \end{cases}$$

Elementary penetrance can be measured and manipulated computationally.

The system begins with **G** a set of breadth-first final state graphs and utilizes elementary penetrance values directly to estimate (tentatively) the true penetrance of states which arise in future solution attempts. Later on, a plan uses the elementary values indirectly in a complex way for revision of former estimates. There are serious error sources in this procedure, however; for example the

[1] An alternative choice might be to use (g+1)/(t+1) to estimate U.

magnitudes of the count functions play a part, and, especially, the early generalization is made on the basis of easy problem instances. We shall examine these errors after we consider the distribution of the penetrance and the form of the modelled evaluation function.

<u>Example 3.2</u>

Consider our standard example which has an attribute space of four dimensions (see section 2.2). The particular results which we view now resulted from the breadth-first search of four easy puzzle instances, each nine moves away from the goal. Three clusters were formed, whose corner points, count functions and elementary penetrance values are shown in table 3.2.[1]

Table 3.2. Cluster Statistics

corner points		counts		elementary
<u>lp</u>	<u>up</u>	g	t	penetrance[2]
(1,0,0,0)	(5,0,0,3)	20	79	0.25
(6,0,0,0)	(6,0,0,4)	0	31	0.016
(7,0,0,0)	(17,2,0,3)	0	2641	0.0002

[1] The mechanism for cluster creation is given in chapter five. The rectangle set covers only those points encountered.

[2] Zval = 0.5.

3.2.2. DISTRIBUTION OF PENETRANCE

Consider again the population of breadth-first final state graphs G_a (the exhaustive fsg set), and let G be a subset of G_a. For any rectangle r, $U(G,r)$ is a random variable over $\{G\}$ whose distribution cannot be determined directly for interesting problems. In any case, our aim is to provide a system which requires a mimimum of user-specified knowledge about a particular problem, so a standard treatment is desired. Unchanging _first order_ structures facilitate mechanization of the _second order_ plan.

Proportions such as our penetrance are commonly altered by one of two similar transformations, the _logit_ or the _probit_; the resulting random variable has a distribution which is likely closer to normal.[1] Here we consider the former; if u is a penetrance, then $\log (u/(1-u))$ is its logit.[2] In our case, nearly all rectangles encountered in practice have very low penetrance. Thus if the logit is approximated by the natural logarithm, the error introduced is generally insignificant.[3] We use the simplifying assumption that any penetrance has a log-normal distribution (and use a (linear) combination of features to predict the

[1] See Bartlett (1947).

[2] The probit is essentially the inverse of the cumulative normal distribution.

[3] Again, some alternate treatment might be selected instead.

logarithm of the penetrance). This choice allows easy computational combination of various error factors (described in later chapters). The conception of the system anticipated that only rough estimates are required, particularly since feedback seems to provide resilience, so if our assumption regarding penetrance distribution is somewhat inaccurate, the consequences should not be catastrophic.

We assume that the distribution of $W(G,r)$ is the same as that of $U(G,r)$.

3.3. EVALUATION FUNCTIONS

Our evaluation functions are composed of two parts, the grouped component ν, and the modelled component ρ. Both map the set of states into the ranking space $[0,1]$ through the feature vector $\underline{f} = (f_1, f_2, \ldots f_n)$. Both estimate true penetrance. (How they are combined is explained in chapter four.)

The grouped component relates to penetrance estimates of rectangles; the penetrance of a state is simply the penetrance of the rectangle into which it maps (this is formalized in chapter four). The rectangles used are generally not degenerate, although we define the <u>true penetrance of a state A</u> to be the true penetrance of the point rectangle r such that $\underline{f}(A) \in r$.

As for the modelled component, we have already constrained the form to be log-linear, i.e. if $\underline{f} = (f_1, f_2, \ldots, f_n)$ is a feature vector and $\underline{b} = (b_0, b_1, b_2, \ldots, b_n)^T$ is a parameter vector, then $\rho = \exp[\underline{f}'.\underline{b}]$.[1,2] (We can consider $\underline{b}$ to be normalized so that $\rho(A) \leq 1$.) Hence the range of ρ is the probabilistic ranking space $[0,1]$.

Now that we have decided on a form for modelled evaluation functions, we should consider how to arrive at an ultimate one, ρ. Ideally, we might proceed: Calculate the true penetrance values $U(\mathbf{G}_a, r)$ for every degenerate rectangle r (where $\mathbf{G}_a$ is the exhaustive fsg set). Use the points $\underline{lp}(r) = \underline{up}(r)$ to provide n independent variables; use $U(\mathbf{G}_a, r)$ as corresponding response observations for ρ; and perform a log-linear regression. If the (column) parameter vector β results, we have $\rho = \exp[\underline{f}'.\beta]$. This is the <u>ideal</u> (<u>modelled</u>) <u>evaluation</u> <u>function</u>. The system's plan estimates ρ. The approximations are modelled evaluation functions that are improved iteratively (see subsection 2.1.3) using subsets of $\mathbf{G}_a$ and complex estimates of $U(\mathbf{G}_a, r)$ (described in chapters four to six).

1 Recall that the prime indicates an augmented vector formed by the addition of a "1" as the leading element.

2 This general form is necessary somewhat arbitrary, since features can be anything, but we do want an unchanging one to facilitate mechanization, and to minimize the information required from the user for any particular problem. A possible, again standardized extension is proposed in chapter eight.

The approximating technique involves finding elementary penetrance values W(**G**,r) for a small set of final state graphs **G** in certain rectangles r. The center points of these rectangles supply a set of independent variables (attributes) while the logarithms of the elementary penetrance values (specially adjusted to estimate true ones) determine the corresponding response, for a weighted regression which finds an estimate $\underline{b}$ of $\underline{\beta}$, for $\log[\rho] = \underline{f}'\cdot\underline{b}$. (The weights depend on error estimates -- small errors mean large weights.)

3.4. ERROR ESTIMATION

We have an expression for the elementary penetrance of a rectangle, but we would also like a measurement of the reliability of this value as an estimator of the true penetrance in the first iteration (see section 1.6). In addition, we require error estimates for least squares fitting of the modelled evaluation function (chapter four), for purposes of clustering (chapter five), and for revision of estimates of the true penetrance (chapter six). In clustering, for example, the error of a penetrance plays a part because it is precisely penetrance values (of rectangles) that are being grouped. We want to know fairly certainly whether two penetrance values are dissimilar, and the larger the error, the more disparate the penetrance values must be before such an assertion can be made (and a split performed).

Let $\underline{f}$ be a feature vector. Suppose we have some final state graph set $\mathbf{G}$, and rectangle r, with counts $g(\underline{f},\mathbf{G},r)$ and $t(\underline{f},\mathbf{G},r)$. Let us abbreviate these values by g and t, respectively.

There are two sources of error in the penetrance $U(\mathbf{G},r) = g/t$. One relates to the magnitudes of g and t. For example, 1/20 is less reliable than 10/200.

The other error source is one about which we know little. For the first iteration, the search is breadth-first, so if the graph traverser is to have any success, the training set must generally be composed of easy problem instances, not randomly selected. Hence the final state graphs that result are not random. Unfortunately we have no reliable way to estimate the bias that results in $W(\mathbf{G},r)$ from the (tentative) assumption that they are.

This predicament is perhaps not as serious as it appears at this stage, however, for the system is designed so that feedback tends to correct earlier errors. This will become apparent later on in our development.

Let us begin with an error estimate based on the first mentioned source, that which relates to the magnitude of the count values. Because of the system design, and especially in view of the existence of the second kind of error, we need only an approximation.

3.4.1. COUNT ERROR

We have $g = g(G,r)$ and $t = t(G,r)$ while $G = G(\theta,P)$ (θ being an evaluation function and P a problem instance set). Let us assume that the nodes which contributed to the total count t were chosen randomly from the exhaustive fsg set G_a, rather than by θ and P. Then, for a given rectangle r, t is a random variable over $\{G\}$ which has essentially a binomial distribution. Let N be the number of states A in G_a such that $\underline{f}(A) \in r$. Thus the variance of t is Npq, where p is the probability of t and $q = 1-p$. We can estimate p by t/N, so the estimated standard deviation s is

$$\sqrt{N \frac{t}{N} \left(1 - \frac{t}{N}\right)} .$$

If N is large compared with t, the expression becomes approximately $\sqrt{t}$. A similar argument gives an estimate of the estimated standard deviation of the good count g, as $\sqrt{g}$.

Since we have assumed a log-normal distribution for penetrance, it follows that the standard deviation of the logarithm of a penetrance will translate into a deviation <u>factor</u> for the penetrance itself. So let us express an estimated deviation in this manner. If the good count and total count are altered by one standard deviation estimate in the expression for the penetrance, and if this is divided by the original penetrance g/t, we obtain

$$\frac{g+\sqrt{g}}{t-\sqrt{t}} \Big/ \frac{g}{t} = \frac{1+\frac{1}{\sqrt{g}}}{1-\frac{1}{\sqrt{t}}}, \quad \text{if} \quad g>0, \quad t>1.$$

So we define the count deviation factor (df) (of the pair (g,t))

$$\text{devc}(g,t) \underset{\text{def}}{=} \frac{1 + 1/\sqrt{g'}}{1 - 1/\sqrt{t}}$$

where $g' = g$ if $g > 0$ and $g' = \text{zval}$ if $g = 0$, and $t > 1$. The logarithmic count deviation (of g, t) is lndevc(g,t) = log[devc(g,t)]. Note that the lower limit of lndevc is zero, and that of devc is one (both imply no error).

3.4.2. NON-RANDOMNESS ERROR

Now let us consider the other source of error in the elementary penetrance $W(G,r)$ as an estimator of the true penetrance. (This occurs in the first iteration when the evalution function is θ_0 = constant but the training set P_1 is not random; instead it is composed of easy problem instances.) We can only make a rough guess of the effects, but when the search is breadth-first, the graph traverser wanders around aimlessly in the attribute space, beginning with the starting state. Many of the nodes developed are perhaps "worse" than the starting state (i.e. farther from the goal than it is), and they therefore have no chance to lead to a solution, since better nodes are also created at the same level in the state graph. Nevertheless these poorer nodes are developed indiscriminately, along with the good ones. Consequently, $U(G,r)$ may be as small as zero for rectangles r that enclose mainly these relatively poorer nodes. If, on the other hand, the starting state had been farther from the goal, some nodes similar to these "poor" ones may actually have been in a solution; i.e $U(G_\alpha,r)$ may be significantly different from $U(G,r)$, or from $W(G,r)$.

In an attempt to quantify this error, <u>the user-specified deviation factor</u> (<u>df</u>) <u>devu</u> is used. For all our experiments, devu was selected to be a function of the counts: $devu(g,t) = 1 + 1/\sqrt{50max(zval,g)/t}$. (The constant '50' was chosen arbitrarily -- its effect is shown in later examples.) So low elementary penetrance means high error.

The <u>logarithmic user-specified deviation</u> is <u>lndevu</u> = log(devu).

In iterations after the first, an extension of devu estimates the non-randomness bias caused by the fact that the evaluation function is non-trivial. The next chapter defines this extension, and proceeds to utilize the various error factors. (These and other error estimates are central in chapters five and six as well.)

4. THE EVALUATION FUNCTION AND THE ITERATIVE SYSTEM

In this chapter, we define the full evaluation function, and in order to do so, we first introduce the concept of a "region", which groups an attribute space rectangle, its penetrance, and the penetrance error. The use of the region also facilitates an outline of the entire iterative process, given toward the end of the chapter.

4.1. REGIONS

We shall find it useful to express regions in two forms, "reduced" and "unreduced".

4.1.1. UNREDUCED REGIONS

Suppose we are given some integer (attribute) space **A**, and also a final state graph set **G**. Define an *unreduced region* R *(in* **A***)* to be a five-tuple $(r(R), \gamma(R), \tau(R), \mathbf{k}(R), \mathbf{e}(R))$, where $r(R)$ is the *rectangle of* R *(in* **A***)*, $\gamma(R) = g(\mathbf{G}, r(R))$ is the *good count of* R *(for* **G***)*, $\tau(R) = t(\mathbf{G}, r(R))$ the *total count of* R *(for* **G***)*, $\mathbf{k}(R)$ the *multiplier of* R, and $\mathbf{e}(R)$ the *multiplier deviation factor (df) of* R.

The meanings of the last two elements, $\mathbf{k}(R)$ and $\mathbf{e}(R)$ will not really be clear until chapter six, but we can say now that they relate to the past history of a region. $\mathbf{k}(R)$

is a multiplier for $\gamma(R) / \tau(R)$, and will be used so that the expression $\gamma(R) / \tau(R) \cdot \mathbf{k}(R)$ estimates the true penetrance. We shall see later on that generally $\mathbf{k}(R) \neq 1$ when the count functions are applied to fsg sets $\mathbf{G} = \mathbf{G}(\theta,\mathbf{P})$ such that $\theta \neq$ constant.

We define a function over the set of unreduced regions in an attribute space: If R is an unreduced region, then the <u>penetrance of R</u>

$$\underline{\text{val}}(R) \underset{\text{def}}{=\!=} \begin{cases} \gamma(R) / \tau(R) \cdot \mathbf{k}(R), & \text{if } \gamma(R) > 0 \\ \text{zval} / \tau(R) \cdot \mathbf{k}(R), & \text{if } \gamma(R) = 0. \end{cases}$$

<u>4.1.2. ERROR OF A REGION</u>

Let $R = (r,\gamma,\tau,\mathbf{k},\mathbf{e})$ be an unreduced region. Suppose, first, that R has been obtained from a fsg set $\mathbf{G}_1$ in the first iteration, i.e. $\mathbf{G}_1 = \mathbf{G}(\theta_0,\mathbf{P}_1)$ where θ_0 = constant, while $\gamma = g(\mathbf{G}_1,r)$ and $\tau = t(\mathbf{G}_1,r)$. Since the search was breadth-first, the penetrance val(R) is considered to estimate the true penetrance if $\mathbf{k}$ is assigned the value one.

At the end of the last chapter, we defined devu(γ,τ) to be $1 + 1/\sqrt{50\max(\text{zval},\gamma)/\tau}$. In the case of the first iteration, devu is an estimate of the non-randomness error caused by non-representative (easy) problem instances. We can now extend devu to estimate the non-randomness error in iterations after the first, when the search strategy is non-trivial (and problem instances may still be non-representative). Since generally $k \neq 1$ in these cases, if we wish to relate devu to val(R), it is necessary to alter the definition of this deviation factor. Let us use the (arbitrary) expression $1 + 1/\sqrt{50\mathrm{val}(R)}$ (which is simply our original estimate if $k = 1$). To simplify notation, we retain the name "devu" and define _the non-randomness deviation factor (df) of a region_ R to be

$$\underline{\mathrm{devu}}(R) \underset{\mathrm{def}}{=\!=} 1 + 1 \Big/ \sqrt{50\ \mathrm{val}(R)}\ .$$

This can sometimes be quite large. For example, if val(R) = 2×10^{-6} (a not unreasonable value), devu(R) = 101.

Another function, the (_total_) _deviation factor_ (_df_) _of an unreduced region_ R = $(r(R),\ \gamma(R),\ \tau(R),\ k(R),\ e(R))$ is

$$\underline{\mathrm{dev}}(R) \underset{\mathrm{def}}{=\!=} \mathrm{devc}(\gamma(R), \tau(R)) \cdot \mathrm{devu}(R) \cdot e(R)\ ,$$

where devc is the count deviation factor. The _logarithmic total deviation of R_ is _lndev_(R) = log[dev(R)].

4.1.3. REDUCED REGIONS

If R is an unreduced region, then the _reduced region_ (_of R_) is the triple

$$\underline{\mathrm{red}}(R) \underset{\mathrm{def}}{=\!=} (r(R),\ \mathrm{val}(R),\ \mathrm{dev}(R)).$$

The _penetrance_ _val(Q)_ _of a reduced region Q_ is its second element and the _total deviation factor dev(Q) of a reduced region Q_ is its third element.

When there is no danger of ambiguity, we refer to red(R) simply as R.

Example 4.1

Example 3.2 showed three different rectangles and their counts. In region formulation with unity multipliers and unity multiplier df's, these are:

$R_1 = (r_1, 20, 79, 1, 1)$; $R_2 = (r_2, 0, 31, 1, 1)$; $R_3 = (r_3, 0, 2641, 1, 1)$.

Their penetrance values and total deviation factors (if devu = 1) are:

$\text{val}(R_1) = 0.25$	$\text{dev}(R_1) = 1.38$
$\text{val}(R_2) = 0.016$	$\text{dev}(R_2) = 2.94$
$\text{val}(R_3) = 0.0002$	$\text{dev}(R_3) = 2.46$

So the regions, reduced, are simply: $\text{red}(R_1) = (r_1, 0.25, 1.38)$; $\text{red}(R_2) = (r_2, 0.016, 2.94)$; $\text{red}(R_3) = (r_3, 0.0002, 2.46)$.

If devu(R) is defined as $1 + 1/\sqrt{50\text{val}(R)}$ then the regions become: $(r_1, 0.25, 1.77)$; $(r_2, 0.016, 6.2)$; $(r_3, 0.0002, 27.)$. As desired, the deviation factors are higher for low penetrance values, with the use of devu. This permits subsequent revision (discussed in chapter six).

4.2. THE FULL EVALUATION FUNCTION

As we shall see later in this chapter, there is a certain set of regions, which will be termed <u>cumulative</u> regions, that is modified at the second step of each iteration. These special regions contain the entirety of accumulated heuristic information about the state-space problem and attribute space to which they refer, and it is by these regions that we form both components of our evaluation function. Since cumulative regions are altered once for each complete iteration, the evaluation function is dynamic too (it is recreated at the third step of an iteration).

We now examine the reasons for the use of both components, rather than one or the other alone; after which we proceed to define the full evaluation function in detail. (Exactly how the cumulative regions are created and revised is the subject of chapters five and six.)

To repeat, the full penetrance function is composed of two parts (this was earlier mentioned in section 2.1). One, the grouped evaluation function ν, is defined directly from the cumulative regions $\{R\}$. It is piecewise constant; the grouped penetrance of a state A is val(R) if $\underline{f}(A) \in r(R)$. And the other component, the modelled evaluation function ρ, is also a product of the cumulative regions, but indirectly through a regression. ($\rho(A) = \exp[\underline{f}(A) \cdot \underline{b}]$.)

The two components form a synergism. Without the grouped function, the modelled one might always remain a constant. This is because there must be more regions than features, for a meaningful regression, and, generally, little useful information can be gleaned from the initial breadth-first search. In any case, for early iterations, the grouped penetrance ν sometimes discriminates states according to a feature which is not yet in the model. This may allow solution of otherwise unsolvable harder problem instances (and consequently the function can converge in fewer iterations). It is important to note that, in the first iteration or two, there is simply not enough (accumulated) information to generate a good modelled function ρ (recall from the introduction that a feature can possibly not even enter except for harder problem instances).

On the other hand, the modelled penetrance (or some suitable substitute) seems essential. If it is not included, in early iterations not enough discrimination occurs to solve harder problem instances (not enough penetrance categories yet exist), and in later iterations some regions inevitably become "perverted": their penetrance goes awry and the graph traverser repeatedly wanders off in the wrong part of the state-space and is never corrected.

These assertions may be better appreciated if we consider a hypothetical example. Figure 4.2 shows a set of four regions from which an evaluation function might be computed. The grouped component ν has only 4 categories, but in one area, there is differentiation according to f_2. Suppose that there is just a weak discrimination here, i.e. that there is not much difference in the penetrance values v_3 and v_4 (but $v_1 >> v_2$ and $v_2 >> v_3$), so that the modelled component $\rho = \exp[b_0 + b_1 f_1]$ has no f_2 term. Then ρ permits 25 categories in the range [0,24], but all depend on f_1 only. If the two components are combined, 38 categories are available, 26 of which discriminate on the basis of both features together.

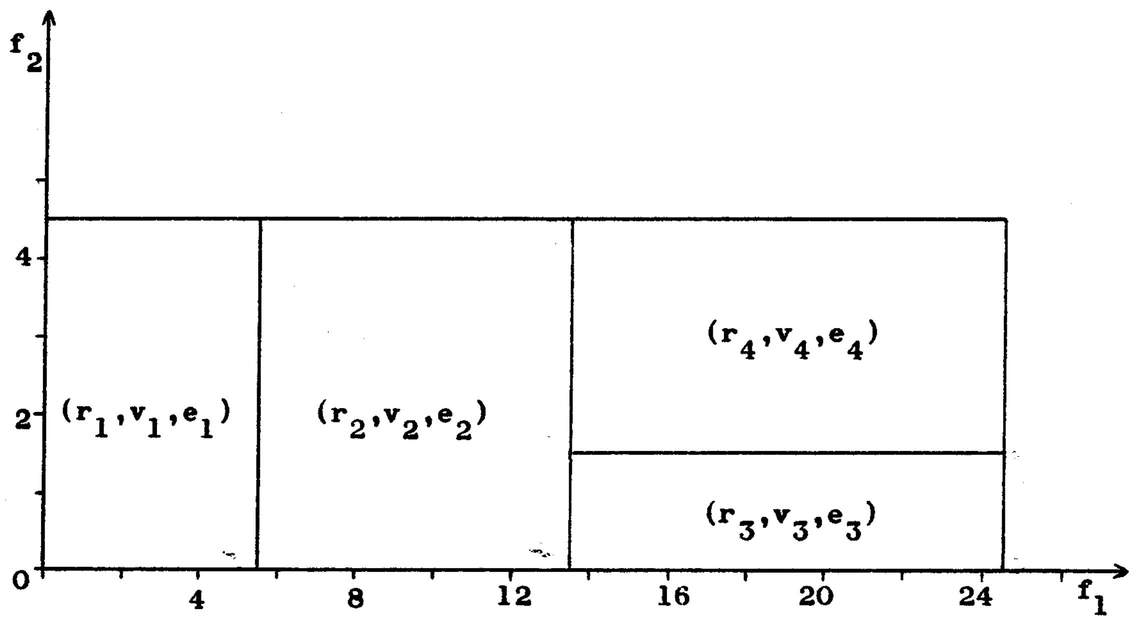


Figure 4.2. Hypothetical Penetrance Function.

Because ρ contains information from all the regions, it is less likely than ν to be erroneous. If the one split in the dimension corresponding to f_2 is realistic, a judicious combination of ν and ρ is an improvement; if this differentiation is anomalous, ρ damps its influence.

Summarizing, the grouped penetrance is generally more sensitive to incipient feature discrimination, and so can "lead" the modelled one, but the latter restrains any malformation in the former. Once a feature is entrenched, the modelled component is more discriminative (i.e. more penetrance categories exist); but, to become established, it initially relies on the grouped one.

4.2.1. GROUPED PENETRANCE

Let us define precisely the direct component. Suppose we have a feature vector $\underline{f} = (f_1, f_2, \ldots, f_n)$, and a set of regions $\mathbf{R} = \{R_1, R_2, \ldots, R_m\}$ in the attribute space $\mathbf{A}$ defined by $\underline{f}$. Let $\underline{x}$ be a point, and r a rectangle, both in $\mathbf{A}$. Denote by $||\underline{x}, r||$ an attribute space distance between $\underline{x}$ and r: $||\underline{x}, r|| = \sum_{i=1}^{n}$ (if $lp_i(r) \leq x_i \leq up_i(r)$ then 0 else $\min[|x_i - lp_i(r)|, |up_i(r) - x_i|]$). We define the <u>grouped penetrance of a state A (for **R**)</u>,

$$\nu(\mathbf{R}, A) \underset{def}{=\!=} \begin{cases} val(R), & \text{if } \underline{f}(A) \in r(R),\ R \in \mathbf{R} \\ \max\limits_{Q \in \mathbf{Q}}(val(Q)), & \text{if } \underline{f}(A) \notin r(R),\ \forall R \in \mathbf{R}, \end{cases}$$

where $\mathbf{Q} = \{Q \in \mathbf{R} \mid \forall R \in \mathbf{R},\ ||\underline{f}(A), r(Q)|| \leq ||\underline{f}(A), r(R)||\}$.

(I.e. use the penetrance of the region closest in the attribute space, or that of the "best" region if there are ties in distance.) If there is no danger of ambiguity in the specification of **R**, we can abbreviate $\nu(\mathbf{R},A)$ to $\nu(A)$.

We also define the <u>grouped</u> <u>penetrance</u> <u>deviation</u> <u>factor</u> (<u>df</u>) (<u>for</u> <u>A</u> <u>and</u> <u>R</u>) in the obvious way:

$$\underline{\text{dev}\nu}(\mathbf{R},A) \underset{def}{=\!=} \text{dev}(R) \ ,$$

where, as above, R is the region whose rectangle surrounds $\underline{f}(A)$, or, failing enclosure, the "best closest" one from **R**. If there is no danger of ambiguity, we can refer to this df as $\text{dev}\nu(A)$.

<u>4.2.2.</u> <u>MODELLED</u> <u>PENETRANCE</u>

In chapter three we decided to construct all modelled evaluation functions so that they estimate the true penetrance. We also decided on a standard form of any such function: $\rho(A) = \exp[\underline{f}'(A)\cdot\underline{\beta}]$ where A is a state, $\underline{\beta}$ is a parameter vector, and $\underline{f}$ is a feature vector. We are now ready to see how to use (cumulative) regions as data to determine the parameter vector. A set of regions **R** specifies the regression information. The rectangle r(R) of a region $R \in \mathbf{R}$ supplies the independent variables -- the center of r(R) is used -- while the logarithm of val(R) (i.e. the logarithm of an estimate of the true penetrance of r(R)) provides the response variable. The logarithmic deviations lndev(R) determine the weighting in the least squares fit.

One purpose of the system is to discover which features are relevant, and of those which are more important; generally this information is not known a priori. Thus, an algorithm which discriminates among and selects variables is appropriate; so the IMSL stepwise regression procedure RLSTEP (subsection 1.6.3) was chosen. The confidence level $1-\alpha$ or uncertainty level α is a parameter of this algorithm.

The details of the regression can now be formalized. We have previously defined the vector functions $\underline{lp}$ and $\underline{up}$ to refer to the extreme corner points of a rectangle (subsection 3.1.1). At this time we need to access the center points, so we define an appropriate vector function. If r is a rectangle, then the center point of r

$$\underline{cp}(r) \overset{}{\underset{def}{=\!=}} [\,\underline{lp}(r) + \underline{up}(r)\,] / 2.$$

Suppose that we have a feature vector $\underline{f} = (f_1, f_2, \ldots, f_n)$ and corresponding space **A**, and a set of regions $\mathbf{R} = \{R_1, R_2, \ldots, R_m\}$ in **A**. Let $\underline{x} = \underline{cp}(r(R_j))$ and $y_j = \log(\mathrm{val}(R_j))$.

The normal equations $X^T V^{-1} X\underline{b} = X^T V^{-1}\underline{y}$ have the solution $\underline{b} = (X^T V^{-1} X)^{-1} X^T V^{-1}\underline{y}$, where X is an m x (n+1) matrix of independent variable observations, $\underline{y}$ is an m x 1 vector of corresponding response observations, $\underline{b}$ is the parameter vector which best fits the data (least squares), and V is the variance matrix of $\underline{y}$ (see Draper & Smith, 1966). To adapt this to our situation, we need an (n+1) x (n+1) <u>selection matrix</u> J which has elements J_{ij} such that $J_{ij} = 0$ if $i \neq j$, $J_{11} = 1$ (constant parameter always included), and, for $i>0$, $J_{i+1,i+1} = 1$ if the ith independent variable is in the model (otherwise $J_{i+1,i+1} = 0$). The normal equations then become $(JX^TV^{-1}X + (I-J))\underline{b} = JX^TV^{-1}\underline{y}$ (where I is the (n+1) x (n+1) identity matrix), and their solution is $\underline{b} = (JX^TV^{-1}X + (I-J))^{-1} JX^TV^{-1}\underline{y}$.

Now we can substitute the values in which we are interested. Let J be the selection matrix determined by RLSTEP. Let X be the m x (n+1) matrix formed with $\underline{x}_j'$ $(1 \leq j \leq m)$ as rows. Let the m elements of $\underline{y}$ be the y_j, and finally, let the non-zero elements of V be $V_{jj} = [\mathrm{lndev}(R_j)]^2$. (In other words, each center point and logarithmic penetrance from a region count an amount which is weighted by the inverse of the logarithm of its total deviation factor.) The parameter vector $\underline{b}$ results (which will have $b_i = 0$ if $J_{i+1,i+1} = 0$). And $\hat{y} = \underline{x}.\underline{b}$ where $\underline{x}$ is a point in the attribute space A, and $\hat{y}$ is the predicted value of the logarithm of the penetrance.

So if A is a state of the space S, the <u>modelled penetrance of A (for region set R and uncertainty level α)</u>

$$\rho(\mathbf{R},\alpha,A) \underset{\text{def}}{=\!=} \exp[\underline{f}(A)'.\underline{b}] \,/\, \max_{B \in S}[\underline{f}'(B).\underline{b}].$$

The divisor normalizes ρ so that $\rho(A) \leq 1$. Hence the range of ρ is the ranking space $[0,1]$. If there is no danger of ambiguity, we can write this as $\rho(\mathbf{R},A)$ or simply as $\rho(A)$.

We shall later have reason to refer to a related function, over a set of regions. If R is a region, then the modelled penetrance of R (for $\mathbf{R}$ and a) is

$$\rho\mathrm{val}(\mathbf{R},a,R) \underset{\mathrm{def}}{=\!=} \exp[\underline{cp}'(r(R)).\underline{b}] \;/\; \max_{B\in S}[\underline{f}'(B).\underline{b}].$$

We also need an expression for the error. Let X^* be the matrix obtained by contracting JX, deleting columns which are zeroed (as a result of the corresponding feature's not being in the model -- so the dimensionality is reduced to the trace of J).[1] Define similarly the other starred matrices and vectors in the following. The variance of y for a particular attribute vector $\underline{x}$ is given by $\underline{x}'^*(X^{*T}V^{*-1}X^*)^{-1}\underline{x}'^{*T}\sigma^2$, and σ^2 is estimated by the regression residual mean sum of squares, s^2. So we define the logarithmic modelled penetrance deviation (for $\mathbf{R}$, a and A)

$$\mathrm{lndev}\rho(\mathbf{R},a,A) \underset{\mathrm{def}}{=\!=} \underline{f}'(A)^*(X^{*T}V^{*-1}X^*)^{-1}\underline{f}'(A)^{*T}s^2.$$

(And $\mathrm{dev}\rho \underset{\mathrm{def}}{=\!=} \exp(\mathrm{lndev}\rho)$.) Define the modelled penetrance df of a region ρvaldev (and its logarithm lnρvaldev) in an analogous way.

[1] It is necessary to eliminate the rows consisting of zeroes, since otherwise the matrix $X^TV^{-1}X$ is singular.

4.2.3. THE COMBINATION

Now that we have defined the grouped penetrance and the modelled penetrance of a state, we arrive at the question of how to combine them into a single estimate of the true penetrance. Let us inspect a different but related problem. Suppose that x is a random variable over a population S, and suppose that a sample of size N is drawn from S which results in an estimated mean $\bar{x}$. $\bar{x}$ is a random variable over samples of size N, and it can be shown that its standard deviation is $\sigma/\sqrt{N}$ where σ is the standard deviation of x.

Now let us suppose that two samples are drawn, of sizes N_1 and N_2, resulting in estimates of the mean of x, denoted by $\bar{x}_1$ and $\bar{x}_2$ respectively. The respective standard deviations are $\sigma/\sqrt{N_1}$ and $\sigma/\sqrt{N_2}$. A single estimate of the mean of x could be derived by calculating $(N_1\bar{x}_1 + N_2\bar{x}_2) / (N_1+N_2)$ whose standard deviation is $\sigma/\sqrt{N_1+N_2}$.

Next, let us suppose that the values of N_1 and N_2 have somehow been lost, but nevertheless we do have estimates s_i of $\sigma/\sqrt{N_i}$ $(i=1,2)$. Then we could estimate N_i to be σ^2/s_i^2. If we substitute for N_i in the original expression, we obtain an approximation of the mean of x which is $(\bar{x}_1/s_1^2 + \bar{x}_2/s_2^2) / (1/s_1^2+1/s_2^2)$, and the estimated standard deviation (again substituting for N_i) is $1/\sqrt{1/s_1^2+1/s_2^2}$. The contribution of $\bar{x}_i$ is larger if its error estimate is lower; the error estimate of the combination is less than that of either $\bar{x}_1$ or $\bar{x}_2$ separately.

We now return to our original problem and define quantities anologously. Since the penetrance is assumed to have a log-normal distribution, we weight lorarithms of penetrance values according to logarithmic deviations. But in cases in which the trace of the selection matrix J, tr(J), is greater than the size of the region set **R** (i.e. when not enough data is present for the size of the model), the grouped penetrance is used solely. The <u>(full) penetrance of a state A (for **R** and a)</u>,

$$\theta(\mathbf{R},a,A) \underset{\text{def}}{=\!=} \begin{cases} \exp\left[\dfrac{\dfrac{\log(\nu(A))}{(\mathrm{lndev}\nu(A))^2}+\dfrac{\log(\rho(A))}{(\mathrm{lndev}\rho(A))^2}}{\dfrac{1}{(\mathrm{lndev}\nu(A))^2}+\dfrac{1}{(\mathrm{lndev}\rho(A))^2}}\right] & \text{if } \mathbf{R} \text{ has more than } \mathrm{tr}(J) \text{ elements,} \\ \nu(A), & \text{if } \mathbf{R} \text{ has between one and } \mathrm{tr}(J) \text{ elements,} \\ 0.5, & \text{if } \mathbf{R} = \phi. \end{cases}$$

We can abbreviate $\theta(\mathbf{R},a,A)$ tc $\theta(\mathbf{R},A)$ or to $\theta(A)$ if there is no danger of ambiguity. Clearly, the range of θ is the ranking space [0,1] (since ν and ρ have this range).

We define θval analogously to ρval. If R is a region, then the <u>full penetrance of R (for **R** and a)</u> is

$\theta\mathrm{val}(\mathbf{R},a,R) \underset{\mathrm{def}}{=\!=}$ the weighted average similar to the above expression for θ, only with val replacing ν, ρval replacing ρ, and the error terms appropriately substituted.

The <u>full penetrance deviation factor</u> (<u>df</u>) (<u>for **R**, a and A</u>) is

$$\mathrm{dev}\theta(\mathbf{R},a,A) \underset{\mathrm{def}}{=\!=} \begin{cases} \exp\left[\dfrac{1}{\dfrac{1}{(\mathrm{lndev}\nu(A))^2} + \dfrac{1}{(\mathrm{lndev}\rho(A))^2}}\right] & \text{if } \mathbf{R} \text{ has more than } \mathrm{tr}(J) \text{ members,} \\ \mathrm{dev}\nu(A)\ , & \text{if } \mathbf{R} \text{ has between one and } \mathrm{tr}(J) \text{ members,} \\ \infty, & \text{if } \mathbf{R} = \phi. \end{cases}$$

Define the <u>full (penetrance) deviation factor</u> of a region devθval (and its logarithm, lndevθval) in an analogous way.

4.3. OUTLINE OF ITERATIVE REVISION

In section 2.1 we began a description of the iterative system, detailing the first, or solving step. It was stated that the full evaluation function from the previous iteration, together with the training problem set for the current iteration, determine the final state graph set for the current iteration; i.e. $G_I = G(\theta_{I-1}, P_I)$. Let us now consider the second step (which manipulates regions) and the third step (which is a regression). Although we cannot yet formally describe the details of the creation and alteration of regions, we can define some basic terms and relationships.

As stated in 4.2, the second step of an iteration accepts a special set of regions, called the cumulative (region) set, along with the final state graph set, and computes a new cumulative set. More specifically, the cumulative region set C_{I-1} from iteration I-1, together with the final state graph set G_I of iteration I <u>determine</u> the <u>cumulative region set</u> C_I <u>of iteration I</u>. Let us denote this mapping by C; then we have $C_I = C(C_{I-1}, G_I)$. (Initially, $C_0 = \phi$.) (C_I is also dependent on θ_{I-1} but this relationship is expressed in G_I.) This constitutes the <u>second step of iteration I</u> or the <u>region handling step of iteration I</u>. The complex mechanism for the region handling step is the subject of chapters five and six, although an overview is presented shortly.

The <u>third step of iteration I</u>, or <u>regression step of</u>

<u>iteration</u> <u>I</u> is the calculation of the full evaluation function from the cumulative region set for iteration I (this procedure was detailed in the previous section); so if A is a state, and α an uncertainty level, the <u>full</u> <u>penetrance</u> <u>of</u> <u>iteration</u> <u>I</u> is $\theta_I(\alpha, A) \overset{=}{\underset{def}{=}} \theta(C_I, \alpha, A)$. If there is no danger of ambiguity in the choice of α (which in practice is often fixed throughout the series of iterations) we can write this $\theta_I(A)$. We can say that θ_I and C_I are <u>associated</u>.

Thus for an entire iteration, we have: (1) $G = G(\theta_{I-1}, P_I)$, (2) $C_I = C(C_{I-1}, G_I)$, and (3) $\theta_I = \theta(C_I)$.

We can think of a cumulative region set C_I as the essential set of structures (see subsection 2.1.2) for our graph traverser, since C_I directly determines the grouped penetrance and indirectly determines the modelled penetrance. The mapping C is the plan.

The following fills in some sketchy details of the region handling step in the context of the entire iterative process. Suppose that a feature vector <u>f</u> is given (determining the space A). Let us examine the first iteration of a series. "CLUSTER" (chapter five) generates the first non-empty set of cumulative regions C_1. It does this by beginning with the single region R = (r, g(<u>f</u>,G_1,r), t(<u>f</u>,G_1,r), 1, 1), where r is the smallest rectangle which will surround all the points in corresponding to states in $\mathbf{G}_1 = \mathbf{G}(\theta_0, \mathbf{P}_1)$ (with $\theta_0 = 0.5$). CLUSTER tentatively splits the rectangle r into twc smaller rectangles. Many such tentative splits of r are examined; and the splits occur in each of the attribute space dimensions. The "best" of these is selected, and this best split is made permanent if the two subregions are "dissimilar" enough. The criterion for dissimilarity is based on the counts of the subrectangles for $\mathbf{G}_1$ and on the count errors. If the two subregions become disjoint, the process is repeated with each subregion as the inspected region, and the splitting continues until nc further discrimination can occur. If the feature set is "useful", then there is eventually more than one element in the resulting region set $\mathbf{C}_1$. If $R' \in \mathbf{C}_1$ is a subregicn with rectangle r', then its counts are $g(\mathbf{G}_1, r')$ and $t(\mathbf{G}_1, r')$.

$\mathbf{C}_1$ becomes the cumulative region set for the regression step of iteration one. The new full penetrance function $\theta_1 = \theta(\mathbf{C}_1)$ becomes something other than the constant 0.5: either ν_1 or else a weighted average of ν_1 and ρ_1 if the number of regions in $\mathbf{C}_1$ is large enough to warrant the regression, as we discussed in the last section.

This completes the first iteration in the formation and modification of the full evaluation function. Succeeding steps are a little different. CLUSTER becomes just part of a larger process; it is used in the latter two of a three-stage algorithm, "REVISE".

The first stage revises penetrance values. Suppose $R \in C_{I-1}$. To improve the accuracy of the penetrance estimate, REVISE modifies val(R) (and dev(R)) according to the elementary penetrance $W(G_I, r(R))$ and <u>also</u> according to $\{W(G_I, r(Q)) \mid Q \in C_{I-1}\}$. The elementary penetrance of the other regions must be taken into account because the non-trivial search strategy has altered the true meaning of the count functions, in such a way that their ratio no longer reflects the true penetrance, but rather a veiled penetrance.

REVISE then proceeds to refine each updated region (second stage), using CLUSTER for splitting in a way similar to that in the first iteration, so that, generally, the number of regions gradually increases over a series of iterations.

The third stage involves enlarging the cumulative regions to accommodate any outlying new attribute vectors.

The situation, however, is more complex during later iterations when θ_I is not a constant. In addition to the inherent difficulty surrounding the veiled aspect of new elementary penetrance values, there is a systematic bias in the values, which should be taken into account when two subregions are split. REVISE is the subject of chapter six.

5. THE CLUSTERING PROCESS

In this chapter, we come to the heart of the system, the clustering algorithm. This algorithm decides whether, and how, a region R should be split, using the counts and deviations of subrectangles of R. Our clustering algorithm is in some ways similar to some well known algorithms; splitting is incorporated. In other respects, however, "CLUSTER" is more unusual. The distance is determined by the experience of the system (specifically, through the count functions), and it is non-metric. Another significant feature of our algorithm is that the final number of clusters is not known a priori, but rather is determined by the data. The precise meaning of these assertions will become clearer in what follows.

We begin with the distance function, which will determine whether two regions are "similar" or not, and if not, how "dissimilar" they are.

5.1. DISTANCE

We want regions to represent volumes in an attribute space, each of which has a fairly uniform penetrance. And we desire multiplicity of regions only to represent diversity of penetrance. Since a distance function in clustering is designed to code dissimilarity and since it can determine the extent of clumping, we would like our distance to reflect divergence of penetrance. Thus we would like the distance between two regions to be high if their penetrance values are quite different, especially if their errors are low.

To derive a suitable distance function, let us begin with the confidence interval for estimates of the mean of a normally distributed random variable x. The interval is $\bar{x} - t_{\alpha/2}\, s/\sqrt{N} < \mu < \bar{x} + t_{\alpha/2}\, s/\sqrt{N}$ for uncertainty level α, standard deviation estimate s and sample size N. Let R_1 and R_2 be two regions. We temporarily assume that $U(\mathbf{G}_a, r(R_1)) = U(\mathbf{G}_a, r(R_2))$ (where $\mathbf{G}_a$ is the exhaustive breadth-first fsg set of chapter three). Let u_j be the penetrance $U(\{G\}, r(R_j))$ $(j=1,2)$ for $G \in \mathbf{G}_a$. Since we have assumed a log-normal distribution, $x_j = \log(u_j)$ is a normally distributed random variable over graphs $G \in \mathbf{G}_a$. Let us suppose that $\bar{x}_j = \log(\mathrm{val}(R_j))$ represents the natural logarithm of a penetrance $U(\mathbf{G}, r(R_j))$ for $\mathbf{G} \in \mathbf{G}_a$ of size N. Let s_j be the estimated standard deviation of x_j. We make the additional assumption that $s_j/\sqrt{N}$ is approximated well by $d_j = \mathrm{lndev}(R_j)$ (c.f. subsection 4.2.3).

Then, substituting in the confidence interval, we have $\bar{x}_1 - t^{(1)}_{\alpha/2}\, d_1 < \mu < \bar{x}_1 + t^{(1)}_{\alpha/2}\, d_1$ and $\bar{x}_2 - t^{(2)}_{\alpha/2}\, d_2 < \mu < \bar{x}_2 + t^{(2)}_{\alpha/2}\, d_2$ where μ is the true mean estimated by both $\bar{x}_1$ and $\bar{x}_2$, assuming they were selected from the same population. The superscript for t reflects the fact that t is a function of sample space size N (slowly varying for larger

N), as well as of α, but let us approximate both $t_{\alpha/2}^{(1)}$ and $t_{\alpha/2}^{(2)}$ by the <u>confidence factor c</u>. Henceforth we shall arbitrarily choose a value for c. The quantity 1.5 for c corresponds roughly to the reasonable assignment of 0.13 to α (while larger values mean smaller α -- i.e. larger c implies greater assurance). Combining the two confidence intervals, we now have $\bar{x}_1 - cd_1 < \bar{x}_2 + cd_2$, or $(\bar{x}_1-\bar{x}_2) - c(d_1+d_2) < 0$.

Suppose $x_1 \geq x_2$. On the basis of the above simplifications and deductions, we would expect that if $E = (\bar{x}_1-\bar{x}_2) - c(d_1+d_2)$ is in fact negative, then $r(R_1)$ and $r(R_2)$ are from the same penetrance population; but the more positive E is, the more sure we would be that $r(R_1)$ and $r(R_2)$ determine disparate true values. Thus E is a reasonable expression with which to measure penetrance disagreement. It can be rewritten $E = (\bar{x}_1 - cd_1) - (\bar{x}_2 + cd_2)$, which is the difference between the "lowest likely" logarithmic higher penetrance x_1 and the "highest likely" logarithmic lower penetrance x_2. From E we derive our distance function.

Suppose we are given two unreduced regions, $R_1 = (r_1, \gamma_1, \tau_1, 1, 1)$, and $R_2 = (r_2, \gamma_2, \tau_2, 1, 1)$, and suppose that $\mathrm{val}(R_1) \geq \mathrm{val}(R_2)$. (Note that the multipliers are exactly one for both regions, as are their deviations -- this is the only case required -- see CLUSTER in the next section.) We define the <u>distance between R_1 and R_2</u> to be

$$\text{dist}(R_1,R_2) \underset{\text{def}}{=\!=} \begin{cases} -\infty, & \text{if } \gamma_j=0 \ (j=1,2) \\ \log[\text{val}(R_1)] - \log[\text{val}(R_2)] + \text{bias} & \\ - c[\text{lndevc}(R_1) + \text{lndevc}(R_2) + \text{biasdev}] & \\ & \text{otherwise.} \end{cases}$$

If $\text{val}(R_1) < \text{val}(R_2)$, $\text{dist}(R_1,R_2) \underset{\text{def}}{=\!=} \text{dist}(R_1,R_2)$.

Bias and biasdev are two terms which will not be fully defined until chapter six. For iteration one, however, and in fact whenever the full penetrance function $\theta_I = \nu_I$ (i.e. when it does not include the modelled component), bias and biasdev are both exactly zero. In defining dist, we have used the count error rather than the total one (subsection 4.1.2 defined the deviation factor of a region to be the product devc . devu . e). This is because, in this case, the non-randomness error, often estimated by devu, is instead taken to be exp(biasdev), and because the multiplier deviation factors e are unity whenever dist is used.

If $\text{dist}(R_1,R_2) \leq 0$, we say that R_1 and R_2 are <u>similar</u>; otherwise they are <u>dissimilar</u>. Obviously, the larger the sum of the logarithmic deviations of R_1 and R_2, and the larger the confidence factor c, the greater the ratio $\text{val}(R_1)/\text{val}(R_2)$ must be in order to cause R_1 and R_2 to be dissimilar.

Note that dist is "cautious", since it implies dissimilarity only if the two penetrance values are disparate enough to overcome their error sum, which may overweight the distance.

Example 5.1

In example 4.1, we calculated the penetrance values and count deviations for three regions $R_1 = (r_1, 20, 79, 1, 1)$, $R_2 = (r_2, 0, 31, 1, 1)$, and $R_3 = (r_3, 0, 2641, 1, 1)$ to be 0.25, 0.016, 0.0002, and 1.38, 2.94, 2.46, respectively. Suppose the confidence factor c = 1.5 and bias = biasdev = 0; then $\mathrm{dist}(R_1,R_2) = \log(0.25) - \log(0.016) - 1.5[\log(1.38) + \log(2.94)] = +0.65$, so R_1 and R_2 are dissimilar. A diagram showing this and the other relationships is given in figure 5.1, where $\log[\mathrm{val}(R_j)]$ is a central value and $c.\mathrm{lndevc}(R_j)$ gives an "error band". The diagram does not reflect the distance between R_2 and R_3; since $\gamma(R_2) = \gamma(R_3) = 0$, $\mathrm{dist}(R_2,R_3) = -\infty$.

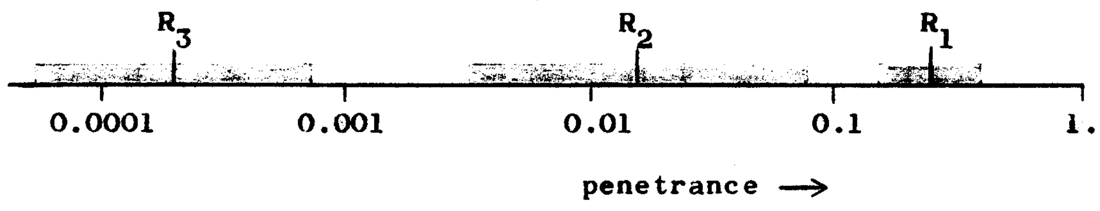


Figure 5.1. Penetrance and Error Bands.

5.2. CLUSTERING

Our algorithm tentatively and variously splits regions R into two, R_1 and R_2 (in one attribute space dimension at a time) and retains the split if $\text{dist}(R_1, R_2) > 0$ and if $\text{dist}(R_1, R_2)$ is a maximum of all possible splits (i.e. if R_1 and R_2 are the most assuredly dissimilar with regard to penetrance). The algorithm continues splitting the subregions until no longer warranted by the data.

5.2.1. THE ALGORITHM

Suppose we are given a feature vector $\underline{f} = (f_1, f_2, \ldots, f_n)$, a set **G** of final state graphs, and a region R_0, called the parent region. In the algorithm below, $\underline{v}_i$ represents the i-th basis vector (with a "1" in the ith position and zeroes elsewhere).

Span is the length of a subrectangle in the direction along $\underline{v}_i$ (fixed for each i). As the algorithm shows, the positive integer parameter, maxspansteps, can possibly increase span, thereby limiting the number of tentative boundaries (successively) inserted. It is designed to speed up CLUSTER in cases in which some features have a very large number of values; in practice it can usually be ∞ (which implies no restriction; i.e. span = 1).

Mintau is another limiting positive integer parameter; it is the minimum total count allowed for a prospective subregion. Throughout all experiments, its value has been 20 (this provides a greater assurance that few anomalous samples arise).

Two "temporary" rectangles, r_1 and r_2, are used to record the extreme corner points of tentative splits. Another, r', retains the corner points of one of the two parts of the best split encountered.

Note that in CLUSTER, the multiplier deviation factors of the temporary regions R_1 and R_2 are unity when dist is used, because the error of the parent region has nothing to do with the splits. The multipliers are also unity at this point. Since only the counts determine divergence of penetrance, the multiplier values are irrelevant.

(The algorithm is on the page following.)

We call the partitioned set, CLUSTER($\underline{f}$,**G**,R_0), the <u>output set of CLUSTER</u> (<u>for parent region</u> R_0, <u>feature vector</u> $\underline{f}$ <u>and final state graph set</u> $\underline{\mathbf{G}}$). If the output set has more than one member, we can say that the feature vector $\underline{f}$ is <u>useful</u> (<u>within</u> $\underline{R}$ <u>for</u> $\underline{\mathbf{G}}$). Similarly the particular features of $\underline{f}$ in whose dimensions splitting occurred, are <u>useful</u>. Useful features <u>discriminate</u> (<u>among states</u>) (<u>according to</u> **G**).

If there is no danger of ambiguity, we can abbreviate CLUSTER($\underline{f}$,**G**,R_0) to CLUSTER(**G**,R_0).

CLUSTER($\underline{f}$,G,R_0)

```
N  :=  the largest subgraph of G containing no edges,  such
                         that $\underline{f}(A) \in r(R_0)$ for all $A \in G \in$ N;
R  :=  {$R_0$};
while ∃ an unmarked $R \in$ R;  comment find best boundary;
   a := lp(r(R));    b := up(r(R));    bestdist := $-\infty$;
   for i:=1 until n do;
      length  :=  (b-a) . $\underline{v}_i$;
      span  :=  max[1, entier(length / maxspansteps)];
      for k:=1 step span until length - span do;
         lp($r_1$) := a;    up($r_1$)  :=  a + k*span*$\underline{v}_i$;
         lp($r_2$)  :=  a + (k+1)*span*$\underline{v}_i$;    up($r_2$)  := b;
         $R_1$ := ($r_1$, g(N,$r_1$), t(N,$r_1$), 1, 1);
         $R_2$ := ($r_2$, g(N,$r_2$), t(N,$r_2$), 1, 1);
         distance := dist($R_1$,$R_2$);
         if distance > bestdist  and  t(N,$r_j$) > mintau  (j =
            1,2) then  (r' := $r_1$;   bestdist := distance);
   if bestdist > 0 then do;  comment split permanently;
      $R_1$ := (r', g(N,r'), t(N,r'), 1, 1);
      $R_2$ := (r(R)-r', g(N,r(R)-r'), t(N,r(R)-r'), 1, 1);
      R := R ∪ {$R_1$,$R_2$} - {R};
   else mark R;
```

Example 5.2

Let us again consider the standard fifteen puzzle example. The same computer results as presented in example 4.1 are listed again below, but this time in the light of the clustering algorithm.

The final state graph set $\mathbf{G}_1 = \mathbf{G}(\theta_0, \mathbf{P}_1)$ resulted from a breadth-first search ($\theta_0 = 0.5$). $\mathbf{P}_1$ had four problem instances whose goals were all at level nine -- these were the hardest instances solvable with θ_0. The parent region for CLUSTER was $R_0 = (r_0, 20, 2721, 1, 1)$, with $\underline{lp}(r_0) = (1,0,0,0)$ and $\underline{up}(r_0) = (17,2,0,4)$. The output set had three regions R_1, R_2, R_3, which are given in table 5.2. CLUSTER first split R_0 into R_3 and $R' = (r', 20, 110, 1, 1)$ with $\underline{lp}(r') = (1,0,0,0)$ and $\underline{up}(r') = (6,2,0,4)$; then R' was further divided into R_1 and R_2. The distances between R_1, R_2 and R_3 are those indicated in example 5.1.

Table 5.2. CLUSTER Output

j	R_j	$\underline{lp}(r_j)$	$\underline{up}(r_j)$
1	$(r_1, 20, 79, 1, 1)$	(1,0,0,0)	(5,2,0,4)
2	$(r_2, 0, 31, 1, 1)$	(6,0,0,0)	(6,2,0,4)
3	$(r_3, 0, 2641, 1, 1)$	(7,0,0,0)	(17,2,0,4)

5.2.2. PROPERTIES OF THE ALGORITHM

First we can see that the procedure always halts. At the start, the region set **R** is just $\{R_0\}$. The first part of the algorithm (inside the "while") is a double loop, the outer one repeated n times, and the inner one also a finite number of times, since rectangles have a limited extent. This part of the algorithm sets bestdist and r'.

The other segment of CLUSTER either marks R_0 (if bestdist ≤ 0) or else it splits R_0 into two, removes R_0 from **R**, and adds the two subregions R_1 and R_2 such that $r(R_0) = r(R_1) \cup r(R_2) = r' \cup r(R_0) - r'$ (the counts of the subregions also summing to those of the parent). Let P be the number of points p in $r(R_0)$ such that p is in the rectangle of an unmarked region. And let the length of the rectangle of R_0 in dimension i be $d_i = up_i(r(R_0)) - lp_i(r(R_0))$. Then the volume of the rectangle is $V = \prod_{i=1}^{n} d_i$. Initially, $P = V$.

If R_0 is marked, ($P = 0$ and) CLUSTER stops; otherwise the situation is as it was at the outset, except **R** now contains two (unmarked) regions R_1 and R_2. Thereafter, on each pass through the "while" loop, an unmarked region R is selected from **R**, and R is either marked (if bestdist ≤ 0) or split. (On a pass through the "while" loop, any rectangle which is not split is immediately marked (since then bestdist < 0). Note that a degenerate region is always marked, since, in this case, length - span < 0 and thus the "for k:= ..." loop is not executed.) Hence the number N of unmarked regions in **R** either decreases by one (if R is marked) or increases by one (if R is split), on each pass.

Since the union of rectangles of all subregions generated from R_0 is exactly $r(R_0)$ at any time, $N \leq V$. Furthermore, the number P of "unmarked" points decreases each time a region R is marked (by the volume of $r(R)$). Thus CLUSTER halts.

Now let us consider the complexity of the algorithm. Suppose that a feature vector $\underline{f} = (f_1, f_2, \ldots, f_n)$ defines an n-dimensional attribute space **A**, that **G** is a final state graph set, and that R is a parent region, these constituting input for CLUSTER. There are several factors affecting the speed of CLUSTER; they arise from the three nested loops. (The second segment of the outermost loop, which records a permanent split, takes relatively little time and can be ignored.)

Let us consider these factors from the inner loop, outward. In the innermost loop, all other steps take an insignificant amount of time compared with the calculation of the counts, since in practice the number of nodes in **G** is hundreds or thousands, and each node A requires a computation to determine if its attribute vector $\underline{f}(A)$ is enclosed by one of the two tentative subrectangles r_1 and r_2. (The computer representation carries the attribute vector with each individual state.) Let us choose as our unit of complexity the time taken for determination of the validity of $lp_i(r_j) \leq f_i(A) \leq up_i(r_j)$ (for both j=1 and j=2). Then the time required for a single pass through the "for k:= ..." loop is $n[t(N,r(R))]$. (The computer implementation of the count functions actually maps states in G to point regions only once, and thereafter the system ascertains directly whether these point regions are enclosed by subregions of the parent region R. Thus, only points in A need to be tallied, rather than states in G; the latter is generally larger -- this advantage is lessened, however, as the number of features and their range are increased.)

The second factor affecting the speed is the total number of possible tentative boundaries that can be inserted in a rectangle. (This corresponds to both "for" loops.) Let us consider the case for which maxspansteps $= \infty$. In dimension i, the number of boundaries is the length $d_i = up_i(r(R)) - lp_i(r(R))$, so the total number for all dimensions is $\sum_{i=1}^{n} d_i = \sum_{i=1}^{n} [up_i(r(R)) - lp_i(r(R))]$.

The third factor governing speed is the eventual number k of permanent splits, since the whole counting process for various tentative rectangles must be repeated twice more each time a split is made permanent. This factor varies as 2k+1. One upper limit of k can be determined as the maximum number of boundaries that can be inserted. This number is the volume $V = \prod_{i=1}^{n} d_i$. This is generally very much larger than another limit which relates to the total count of r(R) for N. Since every subrectangle must have at least mintau state images, $k \leq t(N,r(R)) / \text{mintau}$ (mintau ≥ 1). In fact k is generally much smaller than even this, since permanent subregions must satisfy the stringent distance criterion for dissimilarity, so that in practice k is usually very small (four is the largest observed). (Actually a large k is desirable, since it means high differentiation.)

Thus the total time that CLUSTER requires for a given parent region R is $T \leq n\,[t(N,r(R))]\,[\sum_{i=1}^{n} d_i]\,[2k + 1]$, but in practice it is never more than $Kn\,[t(N,r(R))]\,[\sum_{i=1}^{n} d_i]$ (where K is a small integer constant). If we write $D = \max(d_i)$, then we have $T \leq KDn^2\,[t(N,r(R))]$. (If maxspansteps $< D$, it can replace D in this expression.) The most important fact is that the number of features n can easily be increased; if each feature has roughly the same range of values, the time increases approximately with the square of n.[1] (If the feature ranges are huge, maxspansteps can limit the number of boundaries in appropriate dimensions, with little practical consequence.)

One interesting fact about CLUSTER is that it can sometimes partition a parent region into subregions such that some are similar to each other. For instance, in example 5.2 (see also example 5.1), the parent region R first split into R' and R_3, then R' split again, into R_1 and R_2. Of course R_1 and R_2 are dissimilar (otherwise the split

1 Experiments indicate that with about ten or twelve features, the region handling step typically still takes less time than the solving step.

would not have occurred), and in this case, so are R_1 and R_3. However, R_2 is similar to R_3. (Because $\mathrm{dist}(R_2,R_3) < 0$, $\mathrm{dist}(R_1,R_2) \leq \mathrm{dist}(R_1,R_3) + \mathrm{dist}(R_2,R_3)$ and $\mathrm{dist}(R_1,R_3) \leq \mathrm{dist}(R_1,R_2) + \mathrm{dist}(R_2,R_3)$ cannot both be valid. Hence, this situation of similar subregions could not arise if the triangle inequality held.) This has a desirable effect, however, since subregions differentiate the attribute space, as long as some pair is dissimilar (and this is necessarily the case).

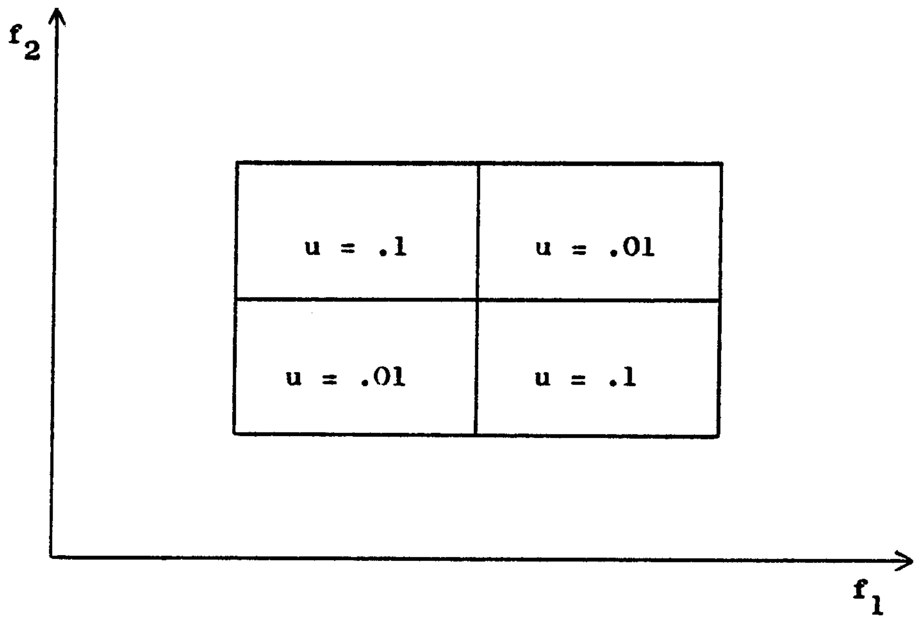


Figure 5.2. Perverse Penetrance Divergence.

It should be noted that some feature sets can possibly confound CLUSTER. For example, suppose that the feature vector is $\underline{f} = (f_1,f_2)$; consider the situation illustrated in figure 5.2 where the elementary penetrance values u = W(G,r) are as shown. In such a (saddle point) situation, certain distributions of point regions would result in no splitting, since CLUSTER observes one dimension at a time.

The problem appears to be more of a theoretical nature, judging from experiments. Even when there are strong interactions amongst features, there seem there seem to be numerous splits. Moreover, the system was designed primarily for reasonably well behaved feature sets.

5.3. SHRINKING REGIONS

In chapter four, the modelled evaluation function was defined in terms of the established regions R, taking the center point of r(R) as the representative attribute vector. This assumes that the unweighted geometric centers are in fact the centers of gravity. We could simply ignore any possibility that r(R) may have an uneven distribution of points (early experiments suggested this would be acceptable); however, our confidence might be greater if the plan were to take account of the point distribution within a cluster. One way of accomplishing this is to shrink regions after splitting. Figure 5.3 illustrates how a more representative center point can result.

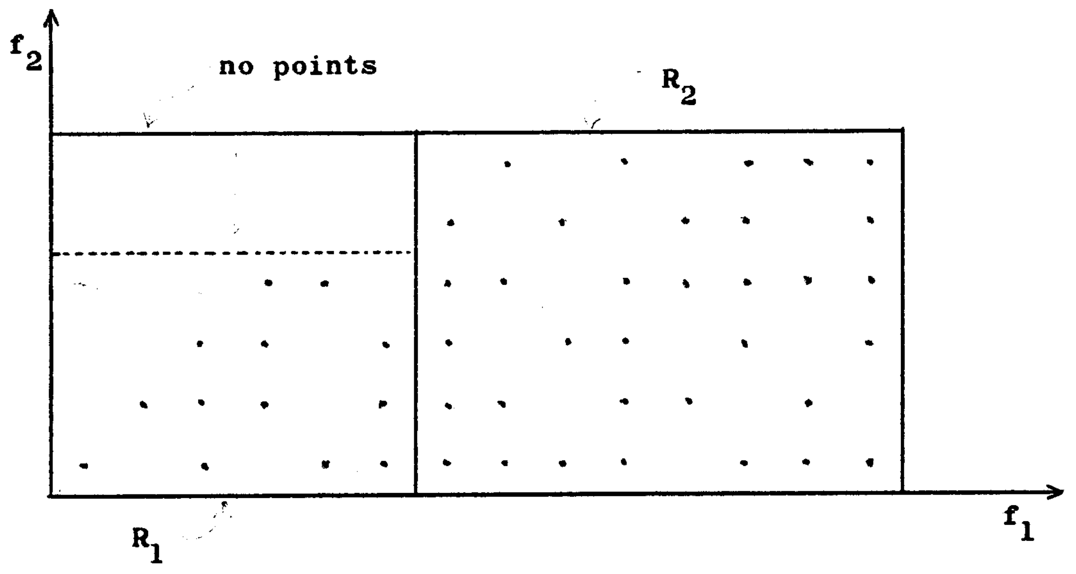


Figure 5.3. Effect of SHRINKing

The following describes such a method, peculiar to this particular system realization (it can be bypassed on first reading). An alternate scheme, which finds the actual center of gravity, is outlined in chapter eight.

Suppose that $\underline{f} = (f_1, f_2, \ldots, f_n)$ is the relevant feature vector, and **R** is a set of (unreduced) regions of cardinality m output by CLUSTER (for $\underline{f}$ and fsg set **G**). We define an algorithm SHRINK, which reduces the sizes of rectangles of regions to enclose minimally the attribute vectors corresponding to **G**, as long as the shrinking does not leave gaps between rectangles (just the peripheral boundaries are affected).

Again, $\underline{v}_i$ is the ith basis vector. The algorithm uses two functions, $\underline{min}$ and $\underline{max}$: $I^n \times I^n \longrightarrow I^n$. If $\underline{a} = (a_1, a_2, \ldots, a_n)$ and $\underline{b} = (b_1, b_2, \ldots, b_n)$, then $\underline{min}(\underline{a}, \underline{b}) = (\min(a_1, b_1), \min(a_2, b_2), \ldots\ldots, \min(a_n, b_n))$. $\underline{Max}$ is defined similarly. SHRINK uses two temporary rectangles r' and r", which it defines by setting their extreme lower and upper points. Finally, the algorithm alters members of the region set $\mathbf{R} = \{R_1, R_2, \ldots, R_m\}$ by (possibly) changing their rectangles. SHRINK accomplishes this by selecting a region $R \in \mathbf{R}$, then defining r' so that it minimally surrounds the points within r(R), and finally testing whether shrinking r(R) to r' (in one dimension at a time) would result in a gap. A gap would result if r", which is a slightly enlarged r(R) (in the testing dimension), intersects some other rectangle r(Q) ($Q \in \mathbf{Q}$). In the first iteration, $\mathbf{Q} = \mathbf{R}$, because no regions other than those in $\mathbf{R}$ are involved. But in post-initial iterations, $\mathbf{Q}$ is a superset of $\mathbf{R}$ — $\mathbf{Q}$ is the whole established set then, not just the parent of $\mathbf{R}$, since gaps are not desired anywhere.

SHRINK(**G**,**R**,**Q**)

```
for j=1 until m do;
    comment find rectangle r' which minimally encloses all
        points in $R_j$;
    S := $\{\underline{f}(A) \in R_j \mid A \in G \in \mathbf{G}\} = \{\underline{p}_1, \underline{p}_2, \ldots, \underline{p}_s\}$;
    $\underline{lp}(r') := \underline{\infty}$;   $\underline{up}(r') := -\underline{\infty}$;
    for k:=1 until s do;
        $\underline{lp}(r') := \underline{min}(\underline{lp}(r'), \underline{p}_k)$;
        $\underline{up}(r') := \underline{max}(\underline{up}(r'), \underline{p}_k)$;

    comment shrink $R_j$ to r' in dimension i if no gaps result;
    $\mathbf{Q}' := \mathbf{Q} - \{R_j\} = \{Q_1, Q_2, \ldots, Q_q\}$;
    for i:=1 until n do;
        $\underline{lp}(r'') := \underline{lp}(r(R_j)) - \underline{v}_i$;   $\underline{up}(r'') := \underline{up}(r(R_j))$;
        for k:=1 until q do;
            if $r'' \cap r(Q_k) \neq \phi$ then go to L;
        $lp_i(r(R_j)) := lp_i(r')$;
        L: (similar statements for upper boundary points)
comment $\{R_1, R_2, \ldots, R_m\}$ is the output region set;
```

The resulting set SHRINK(**G**,**R**,**Q**) is the <u>output</u> <u>set</u> <u>of</u> <u>SHRINK</u> (<u>for</u> **G**, **R** <u>and</u> **Q**).

5.4. THE FIRST ITERATION OF A SERIES

We are now in a position to define exactly what the full evaluation function is after the end of the first iteration. In chapter four we stated that the initial penetrance function θ_0 is always a constant, 0.5. Let $\mathbf{P}_1$ be the training problem set for (step one of) the first iteration. We determine the final state graph set for the first iteration, $\mathbf{G}_1 = \mathbf{G}(\theta_0, \mathbf{P}_1)$. This is the first, or solving step.

The second, or region handling step is as follows. Suppose that the relevant feature vector is $\underline{f} = (f_1, f_2, \ldots, f_n)$. Let A represent a state in $G \in \mathbf{G}_1$. Find the rectangle r which minimally surrounds all $\underline{f}(A)$, for all $A \in G \in \mathbf{G}_1$. Set $R := (r, g(\underline{f}, \mathbf{G}_1, r), t(\underline{f}, \mathbf{G}_1, r), 1, 1)$; R becomes the parent region for the clustering algorithm, as we calculate:

```
$\mathbf{R}$ := CLUSTER($\mathbf{G}_1$,R);
$\mathbf{R}$ := SHRINK($\mathbf{G}_1$,$\mathbf{R}$,$\mathbf{R}$);
$\mathbf{C}_1$ := {red(R) | R $\in$ $\mathbf{R}$};
```

The set $\mathbf{C}_1$ is the (reduced) cumulative region set of iteration one. We can denote this $\mathbf{C}_1 = \mathbf{C}(\theta_0, \mathbf{G}_1)$ where the plan $\mathbf{C}$ represents the above three instruction algorithm (for a first iteration only; post-initial iterations require the complex procedure detailed in chapter six).

It should be noted that for iteration one, all regions of the cumulative set have their multipliers and their deviations exactly one (before reduction). This is because G_1 came from a breadth-first search, so the elementary penetrance values estimate the true ones directly.

Finally, the third, or regression step is to find the full evaluation function of the first iteration $\theta_1 = \theta(C_1)$ (see section 4.2).

<u>Example 5.4</u>

The parent region $R_0 = (r_0, 20, 2721, 1, 1)$ of example 5.2 had the smallest rectangle which could enclose all attribute vectors for the first iteration. (The total number of states developed was 2721.) The output set of CLUSTER was the set of three regions listed in table 5.2. Table 5.4 shows the effect of SHRINK.

Table 5.4. SHRINK Output

j	R_j	$\underline{lp}(r_j)$	$\underline{up}(r_j)$
1	$(r_1, 20, 79, 1, 1)$	(1,0,0,0)	(5,0,0,3)
2	$(r_2, 0, 31, 1, 1)$	(6,0,0,0)	(6,0,0,4)
3	$(r_3, 0, 2641, 1, 1)$	(7,0,0,0)	(17,2,0,3)

6. REGION HANDLING FOR A POST-INITIAL ITERATION

Post-initial iterations are qualitatively different from the first iteration in only one of the three steps. We outlined the three steps of an iteration in section 4.2: The third, or regression step creates the full evaluation function from the cumulative region set (the cumulative, or established region being the basic structure, carrying attributes and corresponding penetrance). The first, or solving step uses a graph traverser, guided by the evaluation function, operating on the problem instance training set, to produce a final state graph set **G**. The second, or region handling step (the plan) utilizes the information in **G** -- specifically the ccunts, from which are computed the elementary penetrance values -- either to establish a region set (first iteration -- see section 5.4), or to update this cumulative set (post-initial iterations), by increasing the amount, and the accuracy, of its information. This chapter is entirely devoted to region alteration in these later iterations.

The region handling step, or plan, is now more complex, however, one of the reasons being that the elementary penetrance values no longer directly approximate the true ones. Nevertheless it is still the elementary values that serve as the essential instruments for region manipulation.

The plan has other necessary complexities, and it incorporates three distinct substeps to handle them all.

The first substep uses the elementary penetrance in an indirect way to revise true penetrance estimates of the established regions (without altering their rectangles).

In addition to updating penetrance values of the established regions, this chapter describes how these old regions can be further subdivided using CLUSTER and the new data. Thus the rectangles are refined -- further discrimination occurs in the attribute space.

Also, sometimes the established region set does not accommodate all of the attribute space maps of the recent final state graph set **G**, so the the rectangles of the old regions are extended to surround all new attributes. After this, another round of possible splitting takes place. (Extension occurs after penetrance revision, rather than before, because penetrance values are adjusted as part of the extension process, and this chosen substep ordering permits greater accuracy since the first substep revises penetrance.)

This chapter is therefore divided into three sections (plus one which sums up the entire process), each of which formalizes the particular substep of the post-initial region handling step: (1) penetrance revision of cumulative regions, (2) region refinement, and (3) extension, and subdivision of extensions.

6.1. REVISING PENETRANCE OF ESTABLISHED REGIONS

First let us consider how we might revise penetrance values of established regions (without disturbing their rectangles). At the end of chapter five it was stated that the elementary penetrance of a rectangle (i.e. the count ratio) becomes its true penetrance estimate for the first iteration. This is because the associated evaluation function is then simply a constant (i.e. the search is breadth-first). But now, for later iterations, the evaluation function is non-trivial, and consequently the situation is more complex, as the following discussion attests.

Consider a breadth-first final state graph G_0 which has resulted from the solution of a problem instance P and constant evaluation function θ_0; and, superimposed on G_0, imagine another fsg G_s which has a solution of P resulting from a best-first search with a non-trivial function θ. For simplicity, suppose the two solutions are identical. If θ is a good evaluation function, the total count $t(\{G_s\},r) < t(\{G_0\},r)$ for most rectangles r, whereas the good count $g(\{G_s\},r) = g(\{G_0\},r)$ for all r. Hence the elementary penetrance $W(\{G_s\},r)$ is typically greater than $W(\{G_0\},r)$. Generalizing this tendency, we expect that, for fsg sets **G**, $u_s = W(\mathbf{G},r)$ is normally larger than the true penetrance $u_0 = U(\mathbf{G}_a,r)$. Since the evaluation functions are non-trivial in the post-initial iterations of our system, the corresponding elementary penetrance values are consequently <u>not</u>

approximations of the true ones.

If we could somehow convert the elementary values to reasonable estimates of the true ones, however, we could use the resulting quantities to revise former cumulative estimates (say, by averaging). In an attempt to realize this goal, we postulate that the elementary penetrance $u_s = u_0^h$, where u_0 is the true penetrance and h is a constant whose value is zero or greater. We call h the <u>strategy-power exponent</u>. This relationship has the property that the elementary penetrance is one whenever the true penetrance is one. Note that, if the evaluation function θ associated with u_s is a constant, then h will be close to one. If θ determines a strategy that is worse than breadth-first, then h will tend to be be greater than one. For normal situations, in which θ is a good function, however, the strategy-power exponent h will be less than one, and the better θ is, the closer h is likely to be to zero. (If θ were a perfect function, h would be identically zero.) For values of h that are intermediate between one and zero, $u_s = u_0^h$ suggests that the evaluation function has more difficulty in areas of low penetrance, which may often be equivalent to the assertion that the function has less success evaluating states which are far from the goal.

In exploratory experiments, when the true penetrance was replaced by estimates from cumulative regions in the above postulated relationship, log-log plots showed a rough linearity (but see chapter eight).

Now the mechanism for the revision of penetrance can be outlined (the details are the topic of the subsections to follow). Use the true penetrance estimates of each established region R, along with the elementary values from the counts of the current fsg set (these being housed in "immediate" regions whose rectangles correspond with the established ones), and estimate h, by performing a log-log-linear regression. Then $u_0 = u_s^{1/h}$. To obtain an improved estimate of the true penetrance, average u_0 with the former true penetrance estimate val(R) of the cumulative region R (weighting according to errors).

6.1.1. FORMALLY RELATING IMMEDIATE TO CUMULATIVE REGIONS

Suppose we have a cumulative region set $\mathbf{R} = \{R_1, R_2, \ldots, R_m\}$, and a resulting full evaluation function θ for $\mathbf{R}$, which has in turn determined a final state graph set $\mathbf{G} = \mathbf{G}(\theta, \mathbf{P})$. Define the (unmodified) immediate region set (for $\mathbf{R}$ over $\mathbf{G}$)

$$\mathbf{I}(\mathbf{R},\mathbf{G}) \underset{\text{def}}{=\!=} \{(r(R),\ g(\mathbf{G}, r(R)),\ t(\mathbf{G}, r(R)),\ 1,\ 1) \mid R \in \mathbf{R}\}.$$

If there is no danger of ambiguity, we can refer to $\mathbf{I}(\mathbf{R},\mathbf{G})$ simply as $\mathbf{I}$.

As just discussed, we postulate that the elementary penetrance $u_s = u_0^h$, where u_0 is the true value, and h is a positive constant. Suppose $R \in \mathbf{R}$, $R' \in \mathbf{I}$, and $r(R) = r(R')$. Then the elementary penetrance $u_s = W(\mathbf{G}, r(R)) = val(R')$, and the best estimate of $u = U(\mathbf{G}_a, r(R))$ is supposedly $\theta val(R)$ (subsection 4.2.3), so we hypothesize that, for all R and R'

$$\log[W(\mathbf{G}, r(R))] = \log(val(R')) = h * \log(\theta val(R)) \qquad (6.1)$$

where h is a constant. (Though the true penetrance has been replaced by an estimate, we retain the "h" for simplicity.)

We can abbreviate $\log(\theta val(R_j))$ to x_j and $\log(val(R_j'))$ to y_j; then we have the model $y = hx$. Now, the reliability of each datum varies, so we weight the values (as in subsection 4.2.2). Let the estimated variance-covariance matrix be given by

$$Vs^2 = \begin{bmatrix} s_1^2 & & & & O \\ & s_2^2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ O & & & & \cdot \quad s_m^2 \end{bmatrix}$$

where s_j is a (logarithmic) standard deviation estimate for R_j' which we detail shortly. If we write the estimate of h simply as h (again for notational simplicity), we have that $h = (\underline{x}^T V^{-1} \underline{x})^{-1} \underline{x}^T V^{-1} \underline{y}$, where $\underline{x}$ and $\underline{y}$ are the m-dimensional vectors composed of the x_j and y_j. This becomes

$$h = \frac{\sum_{j=1}^{m} \frac{x_j y_j}{s_j^2}}{\sum_{j=1}^{m} \frac{x_j^2}{s_j^2}} .$$

The predicted value of y is $\hat{y} = hx$. The variance of h, hvar $= V(h) = (\underline{x}^T V^{-1} \underline{x})^{-1} \sigma^2$. If the model is correct, σ^2 is the variance of the model error $\epsilon_j = y_j - hx_j$. The value of hvar = V(h) is

$$= x_j^2 \frac{\sum_{j=1}^{m} \frac{(y_j - hx_j)^2}{s_j^2}}{\sum_{j=1}^{m} \frac{x_j^2}{s_j^2}}.$$

Now, $x_j = \log(\theta val(R_j))$ and $y_j = \log(val(R_j'))$, and not only y_j, but also x_j has an associated error. The logarithmic deviation in y_j is lndev(R_j') (subsection 4.1.2) and the logarithmic deviation in x_j is lndevθval(R_j) (subsection 4.2.3). It is equivalent to assume no error in x_j and to translate the actual x_j error to a component of the y_j error. Thus we have s_j = lndev(R_j') + h . lndevθval(R_j). Since h is not known initially, the procedure for its calculation, given in the next subsection, is iterative. (This is a fixed point type iteration.)

6.1.2. SETTING MULTIPLIERS OF IMMEDIATE REGIONS: KMOD

Now we are in a position to define the procedure KMOD which essentially applies the regression described in the last subsection, to convert the elementary penetrance of each immediate region to the true penetrance estimate based on equation 6.1. Because of the requirements resulting from the iterative nature of KMOD, the immediate regions are left in unreduced form, which necessitates the computation of their multipliers. Let θ, **R** and **G** be as above, again with **I** = **I(R,G)** the immediate (unreduced) region set over **R** for **G**. Suppose $R \in \mathbf{R}$, $R' \in \mathbf{I}$, with $r(R) = r(R')$. After computing the strategy-power exponent h, KMOD finds the true penetrance estimate $u_0 = u_s^{1/h} = [W(\mathbf{G}, r(R))]^{1/h}$. Thus, to modify $\mathrm{val}(R') = u_s \cdot \mathbf{k}(R')$ to become u_0, $\mathbf{k}(R')$ is simply set to u_0 / u_s.

An estimate for the multiplier deviation factor is also needed. Expressing the quantities logarithmically, we have $\log(u_0) = (1/h)\log(u_s)$. If we allow h to vary by its estimated standard deviation $\sqrt{hvar}$ (and disregard any error in the counts, which is taken into account when the regions are later reduced), we have $\log(u_0) = [1/(h \pm \sqrt{hvar})]\log(u_s)$. If $\sqrt{hvar}/h$ is small, this expression is approximately $[1/h \mp \sqrt{hvar}/h^2]\log(u_s)$. When this is expressed as an error factor (in the non-logarithmic form), it becomes $u_s^{-\sqrt{hvar}/h}$. Combining it multiplicatively with the user-defined df -- which accounts for non-randomness in the counts -- results in the multiplier df shown in the algorithm below.

KMOD$(\theta,\mathbf{R},\mathbf{I})$

```
comment  R = {R_1, R_2, ... , R_m}  &  I = {R_1', R_2', ... , R_m'};
lasth := 0;    h := 1;
while  |h - lasth| / |h|  >  tolerance  do;
    for j:=1 until m do;
        k(R_j') := 1;
        $x_j$ := log(θval(R_j);
        $y_j$ := log[ val(R_j') ];
        $s_j$ := lndev(R_j') + h * lndevθval(R_j);
    lasth := h;
    h := $\sum_{j=1}^{m} (x_j y_j / s_j^2) \,/\, \sum_{j=1}^{m} (x_j^2 / s_j^2)$;
    hvar := $\sum_{j=1}^{m} [ (y_j - h x_j)^2 / s_j^2 ] \,/\, \sum_{j=1}^{m} (x_j^2 / s_j^2)$;
    for j:=1 until m do;
        $u_s$ := val(R_j')   ( = $e^{y_j}$ = W(G,r(R_j)) );
        $u_0$ := exp($y_j$/h);
        k(R_j') := $u_0 / u_s$;
        e(R_j') := $u_s - \sqrt{hvar}/h$ * devu(R_j');
```

The resulting set of regions $\mathbf{M}(\mathbf{R},\mathbf{I})$ = KMOD$(\theta,\mathbf{R},\mathbf{I})$[1] is called the <u>modified immediate (region) set (of I for R)</u>. (The parameter "tolerance" was 0.1 for all experiments.) If

[1] Recall that the cumulative region set $\mathbf{R}$ determines the full evaluation function θ.

there is no danger of ambiguity, we can refer to **M**(**R**,**I**) simply as **M**, as we have done above with h, which is actually also a function of **R** and **I**. We call h the <u>strategy-power exponent</u> (<u>relating **R** to **I**</u>).

6.1.3. <u>COMBINING CUMULATIVE WITH IMMEDIATE REGIONS</u>

Now it is a comparatively simple matter to use the modified immediate regions to update the corresponding cumulative ones -- by taking an average of the two. Since the reliability is different for each of the two members of a rectangle-matching pair, we weight the penetrance values accordingly. As in subsection 4.2.3 we weight logarithmic values by the inverse of their squared logarithmic deviations.

Let **R** = $\{R_1, R_2, \ldots, R_m\}$ be a set of regions. (We define the average for any number of regions, rather than for just two, because we use the general case later in this chapter.) The <u>average penetrance of the regions of **R**</u>

$$\underline{\text{avgval}}(\mathbf{R}) \underset{\text{def}}{=\!=} \exp\left[\frac{\displaystyle\sum_{j=1}^{m} \frac{\log(\text{val}(R_j))}{\text{lndev}^2(R_j)}}{\displaystyle\sum_{j=1}^{m} \frac{1}{\text{lndev}^2(R_j)}}\right].$$

And we have the <u>deviation factor of the average of **R**</u>

$$\underline{\text{devav}}(\mathbf{R}) \overset{\text{def}}{=\!=} \exp \sqrt{1 \Big/ \sum_{j=1}^{m} \frac{1}{\text{lndev}^2(R_j)}} \;.$$

Hence, if R is an established region, and Q is a corresponding mòdified immediate region (i.e. if r(R) = r(Q), with both val(R) and val(Q) representing true penetrance estimates), then the improved estimate is avgval({R,Q}) (and the new deviation factor is devav({R,Q})).

<u>6.1.4. THE PENETRANCE REVISION SUBSTEP</u>

Finally, we are in a position to sum up precisely how immediate regions (which house elementary penetrance values) are used to revise the true penetrance estimates of the cumulative set. Suppose that a cumulative set and its full evaluation function are **R** and θ, respectively, and let **G** be the final state graph set for θ. The penetrance revision algorithm, which constitutes the first substep of the post-initial region handling iterative step, is given below.

```
UTLREV($\mathbf{R}$,$\theta$,$\mathbf{G}$)

Create I, the unmodified region set for $\mathbf{R}$ over $\mathbf{G}$;
$\mathbf{M}$ := KMOD($\theta$,$\mathbf{R}$,$\mathbf{I}$);
$\mathbf{Q}$ := $\{red(R) \mid R \in \mathbf{M}\}$;
$\mathbf{U}$ := $\{ T \mid r(R) = r(Q), R \in \mathbf{R}, Q \in \mathbf{Q}$ and
                  $T = (r(R), avgval(\{R,Q\}), devav(\{R,Q\})) \}$;
```

The first three statements of UTLREV provide a new estimate of the true penetrance of the rectangle of a region, based on its elementary penetrance and equation 6.1, while the final statement averages this estimate with that of the corresponding established region, to give an improved true penetrance estimate.

The set $\mathbf{U}$ = UTLREV($\mathbf{R}$,θ,$\mathbf{G}$) is the <u>penetrance-revised</u> (<u>region</u>) <u>set</u> <u>of</u> $\mathbf{R}$ (<u>for</u> <u>θ</u> <u>and</u> $\mathbf{G}$). If $\mathbf{R}$ is $\mathbf{C}_{I-1}$, the cumulative set of iteration I-1; θ is θ_{I-1}, the full evaluation function of iteration I-1; and $\mathbf{G}$ is $\mathbf{G}_I$, the final state graph set of iteration I; then UTLREV(θ_{I-1}, $\mathbf{C}_{I-1}$, $\mathbf{I}_I$) is the <u>penetrance-revised</u> (<u>region</u>) <u>set</u> <u>of</u> <u>iteration</u> <u>I</u>. A side effect of KMOD is the calculation of h_I = $h(\mathbf{C}_{I-1}, \mathbf{I}_I)$; this <u>strategy-power</u> <u>exponent</u> <u>of</u> <u>iteration</u> <u>I</u> will be used again later.

6.1.5. THE STRATEGY-POWER FACTOR

Let us consider a "local" measure of performance of an evaluation function θ -- within a rectangle r. Suppose that **P** is a problem instance set. Then the associated fsg set $\mathbf{G} = \mathbf{G}(\theta, \mathbf{P})$ determines the elementary penetrance $u_s = W(\mathbf{G}, r)$. We define the <u>strategy-power factor of r (for θ and **P**)</u> to be

$$H(r, \theta, \mathbf{P}) \underset{\text{def}}{=\!=} u_s \,/\, u_0 = W(\mathbf{G}, r) \,/\, U(\mathbf{G}_\alpha, r)$$

(where $u_0 = U(\mathbf{G}_\alpha, r)$ is the true penetrance of r). (If there is no danger of ambiguity, this can be abbreviated to H(r).)

H(r) generally represents the ratio of nodes developed in a typical breadth-first search, to the number developed using the evaluation function θ, for the problem instance set **P**, within the rectangle r; so H(r) is a measure of the performance of θ within r. (Strictly, this interpretation disregards the fact that the good counts can vary from one search to another, but the effect of this inconstancy in the good counts is perhaps not very significant, since solutions might at worst be a few times longer than the shortest, whereas the power factors are frequently very large.) The elementary penetrance can be considered to be an absolute measure of the worth of θ in r, while the power factor H(r) can be thought to be a normalized quantity, a measure of the efficiency of θ in the area r of the attribute space.

In order to obtain an estimate of H(r), the true penetrance could be replaced by (for example) the penetrance of an established region R with rectangle r. If **R** is a cumulative region set associated with the full penetrance function θ, and **G** = **G**(θ,**P**) is an fsg set resulting from the problem instance set **P**, then $W(\mathbf{G},r(R))$ / val(R) (or $W(\mathbf{G},r(R))$ / θval(R)) ($R \in \mathbf{R}$) is an estimate of $H(r(R))$.

It should also be noted that if the true penetrance u_0 is approximated by the penetrance of a region Q output by KMOD (i.e. if Q is a modified immediate region whose penetrance is $u_s^{1/h}$), then the resulting estimate of $H(r(Q))$ is exactly the inverse of the multiplier **k**(Q) of Q.

Since $u_s = u_0^h$, we also have that the strategy-power factor $H(r) = W(\mathbf{G},r) / U(\mathbf{G}_a,r) = u_s/u_0 \doteqdot u_0^h/u_0 = u^{h-1}$. If h=1, $H(r) = 1$ for all r (this would be the case if **G** were from a breadth-first search). If h=0, $u_s = 1$ for all r, which is the ideal.

An illustration of these relationships is included in example 6.1 below.

The stategy-power exponent could conceivably be used (perhaps with a standard problem instance set) to compare the estimates of the power factors based on $H(r) = u_0^{h-1}$, with the estimates obtained from $H(r) = u_s/u_0$, to test the behavior of θ within local rectangles r, and to motivate a search (possibly mechanized) for new useful features in areas of poor discrimination.

<u>Example 6.1</u>

The following is a continuation of our standard fifteen puzzle example. The cumulative set $\mathbf{C}_1$ was $\{ (r_1, 0.25, 1.8), (r_2, 0.016, 6.2), (r_3, 0.00019, 27.) \}$. The training problem set for the second iteration consisted of twelve puzzle instances whose goal levels ranged between twelve and fourteen. (Even the differentiation of the attribute space into only three regions allowed solution of harder problem instances. There is a discussion of this phenomenon in Michie & Ross (1970).) The unmodified immediate region set was $\mathbf{I}_2 = \{ (r_1, 67, 197, 1, 1), (r_2, 19, 106, 1, 1), (r_3, 27, 1783, 1, 1) \}$. Applying KMOD resulted in the computation of the strategy-power exponent h, which became 0.55, and the procedure also gave the modified immediate set $\mathbf{M}_2 = \{ (r_1, 67, 197, 0.41, 1.7), (r_2, 19, 106, 0.25, 2.7), (r_3, 27, 1783, 0.032, 18.) \}$. Reducing the modified set, we obtain: $\mathbf{Q}_2 = \{ (r_1, 0.14, 2.1), (r_2, 0.044, 3.6), (r_3, 0.00049, 22.) \}$. And merging $\mathbf{Q}_2$ with $\mathbf{C}_1$, via avgval, we have the penetrance-revised set, $\mathbf{U}_2 = \{ (r_1, 0.20, 1.6), (r_2, 0.032, 2.8), (r_3, 0.00026, 9.4) \}$. (Notice the magnitudes of the deviation factors are reduced.)

The estimated true penetrance of the second revised cumulative region Q_2 (of $\mathbf{U}_2 = \{Q_1, Q_2, Q\}$) was $u_0 = 0.032 = 1/32$. Since the elementary penetrance $u_s = u_0{}^h$, we would expect the search strategy of the first iteration to have made use of exp(0.55*log(0.032)) = 0.15 of the states mapping to that region, or about one in 7, as opposed to one in 32 for a breadth-first search. Comparing this figure with the corresponding region R_2 of the unmodified immediate set I_2 = $\{R_1, R_2, R_3\}$, we find that about one in 6 was actually used. A similar calculation for the third region pair Q_3, R_3 shows that while only about one in 3800 states would be used in a breadth-first search, the penetrance function θ_1 theoretically improves this to one in 85, and the actual value from the third immediate region R_3 is one in 66. For the first region pair, the values are: one in 4.9 in a breadth-first search, one in 2.4 with θ_1, theorectically; and one in 2.9 with θ_1, actually.

The strategy-power factor estimates (using U_2, I_2 and $H(r) \doteq u_s/u_0$, and the revised true penetrance estimates $val(Q_j)$ for u_0) are 1.7, 5.7, and 59. Using $H(r) = u_0^{h-1}$ gives 2.1, 4.7 and 38.

6.2. SUBDIVIDING ESTABLISHED REGIONS

After the cumulative regions are modified as described in the preceding section, each region of that resulting penetrance-revised set is possibly refined (i.e. the rectangles are allowed to split further). This additional differentiation of the attribute space is highly desirable since it increases the amount of information comprising the evaluation function.

Rectangles can be further subdivided in post-initial iterations because new final state graph data are available after each solving step, and (recall from chapter five that) it is the good versus total counts for the fsg set that determine the distance, which is the essence of CLUSTER.

The continued splitting is basically similar to the clustering that occurs during the first iteration of a series. In the latter, a single all-encompassing region is supplied to CLUSTER (chapter five). In our present case, CLUSTER is called once for each region of the (penetrance-revised) established set, with that region taking the role of the parent region. The algorithm SUBDIV (to be described in this section) is the device which accomplishes this refinement.

But, as discussed in the preceding section, the fact that the evaluation function θ is non-trivial means that the strategy-power factors are not unity (i.e. the elementary penetrance values do not estimate true ones). Thus the multipliers of the newly split regions must be adjusted accordingly (using a suitable operation involving the parent region). Furthermore, if the modelled component of θ is not a constant, the elementary values within a parent region (for CLUSTER) are biased, so the multipliers have to be adjusted for this effect as well. To set the multipliers, SUBDIV calls the algorithm KFIX, which accounts for these two factors in two separate stages.

The distance function (used by CLUSTER) is also affected by the bias, so the full definition of dist is given in this section as well.

6.2.1. ALGORITHMS SUBDIV AND KFIX

Suppose a fsg set **G** has been determined by a full penetrance function θ, and that θ is associated with a cumulative region set **R**. Suppose, also, that **U** = UTLREV(**R**,θ,**G**) = $\{R_1, R_2, \ldots, R_m\}$ is the resulting penetrance-revised set of **R**, h the corresponding strategy-power exponent, and hvar its estimated variance. The second substep in the revision of **R** is a refinement of the regions of **U**, and is defined by the algorithm:

SUBDIV(θ,**G**,h,hvar,**U**)

```
comment  $\mathbf{U} = \{R_1, R_2, \ldots, R_m\}$;
for j:=1 until m do;
    $\mathbf{T}_j$ := CLUSTER($\mathbf{G}$,$R_j$);
    $\mathbf{T}_j$ := SHRINK($\mathbf{G}$,$\mathbf{T}_j$,$\mathbf{U}$);
    KFIX($\theta$,h,hvar,$R_j$,$\mathbf{T}_j$)
$\mathbf{Q} := \bigcup_{j=1}^{m} \mathbf{T}_j$;
$\mathbf{S} := \{S \mid T \in \mathbf{Q}$ & $S = \mathrm{red}(T)\}$;
```

S = SUBDIV(θ,**G**,h,hvar,**U**) is the <u>subdivided</u> (<u>region</u>) <u>set of</u> **U** (<u>for</u> <u>θ</u>, **G**, <u>h</u>, <u>and</u> <u>hvar</u>).

The algorithm KFIX sets the multiplier values for each subregion in $\mathbf{T}_j$. This is essential since the elementary penetrance values $\{u_s = W(\mathbf{G}, r(T) \mid T \in \mathbf{T}_j\}$ do not directly estimate true ones $\{u_0\}$ (recall the relationship $u_s = u_0{}^h$ from the previous section). In our present case, this phenomenon of non-unity power factors has two aspects. Consider one of the parent regions R, with subregions **T** partitioned by CLUSTER. The first aspect concerns the whole set **T** together: If the strategy-power factor H(r(R)) is estimated to be H' = avgval(**T**) / val(R), then the true penetrance of $T \in \mathbf{T}$ can be approximated as W(**G**,r(T)) / H'. Since CLUSTER has set γ(T) to g(**G**,r(T)) and τ(T) to t(**G**,r(T)) (see subsection 5.2.1), the approximation is effected if $k(T) = 1/H'$, for all $T \in \mathbf{T}$.

The second aspect of the disparate penetrance phenomenon has to do with the non-uniformity of power factors _within_ the rectangle r(R) of a parent region R. Within the boundaries of r(R), the (modelled component of the) evaluation function has caused the search strategy to favour the "good" areas of the rectangle, and this has supposedly biased the poorer elementary penetrance values upward. In subsection 6.1.2 we made appropriate adjustment for this bias over the _set_ of cumulative regions; now we must adjust biased penetrance values, _within_ R, in order to find reasonable true penetrance estimates for its new subregions.

If R is the parent region and $T \in \mathbf{T}$ is one of its subregions, the difference of the logarithms of their predicted true penetrance is $\log(\theta val(T)) - \log(\theta val(R))$.[1] From equation 6.1 it follows that the predicted difference in logarithmic elementary penetrance is $h\,[\log(\theta val(T)) - \log(\theta val(R))]$. The difference in these two expressions is what we could call an expected bias in the elementary penetrance. Hence we define

$$\underline{\text{bias}} \underset{def}{=\!=} [1-h]\,[\log(\theta val(T)) - \log(\theta val(R))].$$

Consequently, if h = 1 (which occurs if the search is

1 Note that the modelled component , alone, without , could be substituted for θ, since T and R have the same former grouped evaluation.

breadth-first), bias = 0. Or if the modelled penetrance ρ = 0, bias = 0. Otherwise, the lower the value of h, the higher is the bias. This is what we would expect, according to our previous discussions: we have the elementary penetrance $u_s = u_0^h$, where u_0 is the true penetrance (see 6.1.5). If $u_0^{(1)}$ and $u_0^{(2)}$ are the true penetrances of two rectangles, then the ratio of the corresponding elementary values is $u_s^{(1)} / u_s^{(2)} \doteqdot u_0^{(1)h} / u_0^{(2)h} = (u_0^{(1)} / u_0^{(2)})^h$. This decreases as h does; i.e. the elementary penetrance values appear to indicate less true penetrance disparity as h decreases.

Adding bias to a difference in logarithmic elementary values $d_s = \log[W(G,r(T))] - \log[W(G,r(R))]$ converts d_s to an estimated logarithmic true penetrance difference. To see that this is valid, let us write the logarithmic difference of predicted true penetrance $\log(\theta\text{val}(T)) - \log(\theta\text{val}(R))$ as $\hat{d}_0$ (see the definition of bias, above), and the resulting predicted logarithmic elementary penetrance difference $h[\log(\theta\text{val}(T)) - \log(\theta\text{val}(R))]$ as $\hat{d}_s$. Then bias = $\hat{d}_0 - \hat{d}_s$. Hence d_s + bias = $d_s + \hat{d}_0 - \hat{d}_s = \hat{d}_0$ if $d_s = \hat{d}_s$.

Since a true penetrance estimate is desired for the regions output by SUBDIV, KFIX is designed to alter the multiplier **k**(T) of a subregion T of its parent region R by computing the product of **k**(T) and exp(bias).

In order to approach the error in estimating the multipliers, we can approximate the various contributions separately. For example, let us write the bias $[1-h][\log(\theta val(T)) - \log(\theta val(R))]$ (R being the parent region and T a subregion) as $[1-h].d$, and consider the deviation in the expression due to hvar, the estimated variance of h (neglecting, for the present, any error in d). Allowing h to vary by one standard deviation, this becomes $[(1-h) \pm \sqrt{hvar}]\ d$. Thus the deviation caused by the uncertainty in h is biasdev = $(\sqrt{hvar}).|d|$ = $(\sqrt{hvar} / [1-h])\ .\ |bias|$. As for the other error sources in the calculation of the multiplier of a subregion, we roughly estimate their contribution as a product of the respective deviation factors, so that the entire error estimate is that product times exp[biasdev].

In the following formalization of KFIX, let θ be the full evaluation function, h the appropriate strategy-power exponent and hvar its estimated variance. Let **T** = $\{T_1, T_2, \ldots, T_K\}$ be a set of regions whose rectangles form a partition of the rectangle of a parent region R. KFIX adjusts the multipliers of the regions of **T** in two stages, as suggested above. First any bias within the parent region R is ignored, and the estimate of the inverse strategy-power factor of R is computed, to give a single multiplier value for all subregions of R. Then the bias is taken into account, to alter the subregions individually.

```
                    KFIX($\theta$,h,hvar,R,T)

comment T = {$T_1$,$T_2$, ... ,$T_k$};

comment first convert strategy dependent penetrance of
   immediate set to true penetrance estimates;

for j:=1 until k do;
   k($T_j$)  :=  val(R) / avgval(T) ;
   e($T_j$)  :=  dev(R) * devav(T)

comment correct for any bias;

for j:=1 until k do;
   bias     :=  [1-h] [log($\theta$val($T_j$)) - log($\theta$val(R))];
   biasdev  :=  ( $\sqrt{\text{hvar}}$ / [1-h] ) * |bias|;
   k($T_j$) := exp(bias) * k($T_j$);
   e($T_j$) := e($T_j$) * dev$\theta$val($T_j$) * dev$\theta$val(R) * exp(biasdev);
```

6.2.2. THE FULL DEFINITION OF DISTANCE

At the beginning of chapter five, we defined dist, the distance function for the clustering algorithm. At that time, it was stated that the two terms, bias and biasdev, were zero unless the modelled component of the evaluation function was non-trivial. This bias is exactly what was introduced in the preceding subsection. So we can now fully define the distance (which is necessary since SUBDIV uses CLUSTER).

Suppose that R_1 and R_2 are two regions with val(R_1) $\geq$

val(R_2), and let h_I be the strategy-power exponent of iteration I. Also, let θ_{I-1} be the full evaluation function of iteration I-1. Then the <u>distance between R_1 and R_2 for iteration I</u> is

$$\text{dist}(R_1, R_2, \theta_{I-1}, h_I, \text{hvar}_I) \overset{}{\underset{\text{def}}{=\!=}} \begin{cases} -\infty, & \text{if } g(R_j) = 0 \quad (j=1,2) \\ \log[\text{val}(R_1)/\text{val}(R_2)] + \text{bias}_I \\ \quad -c[\text{lndevc}(R_1) + \text{lndevc}(R_2) + \text{biasdev}_I] & \text{otherwise.} \end{cases}$$

where $\text{bias}_I \underset{\text{def}}{=\!=} [1-h_I] \log[\theta_{I-1}\text{val}(R_1) \,/\, \theta_{I-1}\text{val}(R_2)]$ and

$$\text{biasdev}_I \underset{\text{def}}{=\!=} (\sqrt{\text{hvar}_I} \,/\, [1-h_I]) \cdot |\text{bias}_I|.$$

This addition of the bias term converts the elementary penetrance ratio to the desired true penetrance ratio estimate (see the previous subsection) -- we want dist to measure dissimilarity of <u>true</u> penetrance.

(An examination of CLUSTER (section 5.2) shows that, when dist is called, the multipliers of R_1 and R_2 are one. This does not influence bias or biasdev, since it is a difference in the logarithms of the modelled penetrance values of R_1 and R_2 that determines bias and biasdev; i.e. within the parent region R, a difference in $\log(\theta)$ is equal to the corresponding difference in $\log(\rho)$, since the grouped component ν of the evaluation function θ is constant within R.)

6.2.3. THE REFINEMENT SUBSTEP

If $\mathbf{U}_I$ is the penetrance-revised region set, $\mathbf{G}_I$ is the final state graph set, h_I and $hvar_I$ are the strategy-power exponent and its variance estimate, all of iteration I, and θ_{I-1} is the full penetrance function of iteration I-1, then $\mathbf{S}_I$ = SUBDIV(θ_{I-1}, $\mathbf{G}_I$, h_I, $hvar_I$, $\mathbf{U}_I$) is the _subdivided (region) set of iteration I_.

Summarizing the refinement substep: In order to sophisticate the evaluation function, we wish to differentiate the attribute space further, which is possible since we have a recent fsg set $\mathbf{G}$ that determines new counts, and thus new distances for CLUSTER (which SUBDIV incorporates). But because the evaluation function θ associated with $\mathbf{G}$ is non-trivial, the resulting subregions (partitioned from an established parent region) need non-unity multipliers (because the elementary penetrance values do not approximate true ones). Moreover, even within a parent region, θ has caused non-uniform strategy-power factors -- we considered this phenomenon as a bias in the elementary values. To modify the values of the new split subregions, SUBDIV calls KFIX to set the multipliers -- KFIX allows for both aspects of this bias effect of θ. (We also needed to define dist fully, since CLUSTER relies on this distance function and dist must measure true penetrance disparity.)

Examples 6.2

The first example is from the second iteration, for which the full penetrance function had no modelled component; so this will illustrate only the first part of KFIX (no bias within established regions). One of the penetrance-revised regions of example 6.1 was R = (r, 0.20, 1.6), with lp(r) = (1,0,0,0), up(r) = (5,0,0,3). This, as a parent region for CLUSTER in SUBDIV, spawned two subregions,
R_1 = (r_1,24,24,1,1), lp(r_1)=(1,0,0,0), up(r_1)=(2,0,0,3);
R_2 = (r_2,43,173,1,1), lp(r_2)=(3,0,0,0), up(r_2)=(5,0,0,3). In KFIX, avgval({R_1,R_2}) = 0.42. So the multiplier correction factor was val(R)/0.42 = 0.48. (And the estimated strategy-power factor is 1/0.48 = 2.) Devav({R_1,R_2}) was 1.2, and combining this with dev(R) gave a final deviation factor of 1.9. The two new subdivided regions became (r_1, 24, 24, 0.48, 1.9) and (r_2, 43, 173, 0.48, 1.9), which, when reduced, are (r_1 , 0.48, 2.9) and (r_2, 0.12, 2.4).

For an illustration of the second part of KFIX, we need to consider a case in which the modelled component of the penetrance function is non-trivial. This occurred, for instance, in iteration four of our standard example. The region of the penetrance-revised set which is of concern to us now is R = (r, 0.001, 5.), lp(r) = (7,0,0,2), up(r) =

(10,0,2,5). With this as parent region, CLUSTER output a set whose two members were:

$R_1 = (r_1, 13, 94, 1, 1)$, $\underline{lp}(r_1) = (7,0,0,2)$, $\underline{up}(r_1) = (8,0,2,5)$;

$R_2 = (r_2, 8, 175, 1, 1)$, $\underline{lp}(r_2) = (9,0,0,2)$, $\underline{up}(r_2) = (10,0,2,5)$.

The first part of KFIX changed the multipliers to $val(R)/avgval(\{R_1, R_2\}) = 0.001/0.094 = 0.011$. (Here the power factor is estimated to be $1/0.011 = 90$.) This value was further modified by the bias correction section. We have:

$$\begin{aligned} bias &= [1-h]\,[\log(\theta val(R_j)) - \log(\theta val(R))] \\ &= [1-h]\,[\log(\ val(R_j)) - \log(\rho val(R))] \\ &= [1-h]\,[\underline{cp}'(r(R_j)).\underline{b} - \underline{cp}'(r(R)).\underline{b}]. \end{aligned}$$

The only relevant parameter of the modelled evaluation function was that for the first feature (since this is the only dimension in which splitting occurred), and it was $b_1 = -0.51$. The value of h was 0.25. Thus, for R_1, bias $= (0.75)(7.5-8.5)b_1 = 0.38$. And for R_2, bias $= (0.75)(9.5-8.5)b_1 = -0.38$. Thus the final values for the multipliers were: $\mathbf{k}(R_1) = 0.011*\exp(0.38) = 0.016$ and $\mathbf{k}(R_2) = 0.011*\exp(-0.38) = 0.0073$. The errors will not be calcualated here, but the final penetrance values were $val(R_1) = (0.016)(13/94) = 0.0022$ and $val(R_2) = (0.0073)(8/175) = 0.00036$.

6.3. EXTENDING REGION BOUNDARIES AND FURTHER SUBDIVISION

Especially during earlier iterations, when the problem instances which the system has seen are atypical (easier), the cumulative regions do not generally cover all the attribute points mapped from the new final state graph set. So some of the outer established regions need to be extended to engulf the new points. After that, penetrance values are appropriately adjusted, and further splitting is allowed, similar to that described in the preceeding section.

As do the other two substeps, this third and final substep relies on the counts for the fsg set (from the solving step) to modify penetrance.

First let us examine a means to surround the outlying points.

6.3.1. BOUNDARY EXTENSION

Let $\mathbf{R} = \{R_1, R_2, \ldots, R_m\}$ be a set of regions. Let $\mathbf{G}$ be a final state graph set, and $\underline{f}$ a feature vector. Define the set of states $\mathbf{B} = \{A \in G \in \mathbf{G} \mid \underline{f}(A) \notin R \ \ \forall R \in \mathbf{R}\}$, and the set of attribute vectors $S = \{\underline{p} \mid \underline{p} = \underline{f}(A),\ A \in \mathbf{B}\}$, called the <u>outlying point set for **R** and **G**</u>.

The following procedure extends the regions spatially. It uses the vector functions <u>min</u> and <u>max</u> defined in section 5.3, and the attribute space distance $\| \quad \|$ defined in 4.2.1. ENCLOS finds, for each point, the closest region (in

ENCLOS **(R,G)**

```
R' := R;
S := the outlying point set for R and G;
while $S \neq \phi$ do;
    choose some $p \in S$;
    elim := 0;
    success := false;
    while not success do;
        c := minimum $||p,r(R)||$ such that $R \in$ R' and
                                      $||p,r(R)|| >$ elim;
        Q := $\{R \in$ R' $|\ ||p,r(R)|| = c\} = \{Q_1, Q_2, ..., Q_q\}$;
        overlap := true;    j := 1;
        while overlap and $j \leq q$ do;
            comment temporarily store corner points in $r_Q$;
            lp($r_Q$) := min[p, lp(r($Q_j$))];
            up($r_Q$) := max[p, up(r($Q_j$))];
            if $\exists R \in$ R' such that $r(R) \cap r_Q \neq \phi$ then j:=j+1 else
                comment extend rectangle of region $Q_j$ to be $r_Q$;
                overlap := false;
                $Q_j'$ := ($r_Q$, val($Q_j$), dev($Q_j$));
                R' := R' $\cup \{Q_j'\} - \{Q_j\}$;
        if not overlap then success := true else elim := c;
    S := the outlying point set for R' and G;
```

the attribute space) which can be extended without intersecting another region. It defines a candidate set of regions **Q** whose rectangles have a distance c from a selected point $\underline{p} \in S$. Initially, c is the smallest distance from $\underline{p}$ to the rectangle of any region. If no region of **R** can have its rectangle extended to engulf $\underline{p}$ without intersecting another rectangle, a new candidate set **Q** is defined by setting elim := c, and including regions whose rectangles have a distance just larger than elim.

If **S** is a subdivided cumulative region set, and **G** is a fsg set, then the set **S'** = ENCLOS(**S**,**G**) is the <u>uncorrected extended</u> (<u>subdivided</u>) <u>region set of **S**</u> (<u>for **G**</u>) (uncorrected because the counts for the outlying points have not yet been incorporated into the extensions).

<u>6.3.2. MODIFYING PENETRANCE OF EXTENSIONS</u>

When a subdivided region set has been extended, its old penetrance value may be an inaccurate estimate of the true one for the extension (and recall that the center point of a rectangle is used in the modelled evaluation function). We can utilize the former true penetrance estimates, the corresponding elementary values (of both the extended and unextended rectangles), and the relationship between true and elementary penetrance postulated in section 6.1, to obtain estimates for the true values of the extended rectangles.

Suppose that **S'** is an uncorrected extended set of regions whose unextended counterparts were **S**. Let **G** be a set of final state graphs. Form immediate region sets from both **S** and **S'**: **I** = $\{(r(R),\ g(\mathbf{G},r(R)),\ t(\mathbf{G},r(R)),\ 1,\ 1) \mid R \in \mathbf{S}\}$. And define **I'** , similarly, from **S'**. We would like to alter the penetrance values of the regions of **S'** so that they estimate true ones.

Let θ be the associated full evaluation function, and h and hvar the appropriate strategy-power exponent and its estimated variance. Suppose that $R \in \mathbf{S}$, $R' \in \mathbf{S'}$, $Q \in \mathbf{I}$, and $Q' \in \mathbf{I'}$, with $r(R) = r(Q)$ and $r(R') = r(Q')$, and $r(R) \subseteq r(R')$. Set the reduced region $R'' = (r(R'),\ v,\ e)$, where v and e are true penetrance and deviation factor estimates defined in the following.

We have $v = \mathrm{val}(R'') = \mathrm{val}(R).[\mathrm{val}(R'')/\mathrm{val}(R)] = \mathrm{val}(R).[\theta\mathrm{val}(R'')/\theta\mathrm{val}(R)] = \mathrm{val}(R).[\theta\mathrm{val}(R')/\theta\mathrm{val}(R)]$ (since θval depends only on the rectangle of a region). From equation 6.1 we have $\log[\theta\mathrm{val}(R)] \doteqdot (1/h).\log[\mathrm{val}(Q)]$. So $\theta\mathrm{val}(R) = [\mathrm{val}(Q)]^{1/h}$. Similarly $\theta\mathrm{val}(R') \doteqdot [\mathrm{val}(Q')]^{1/h}$. Substituting in the expression for v, we obtain: $\mathrm{val}(R'') = \mathrm{val}(R).[\mathrm{val}(Q')/\mathrm{val}(Q)]^{1/h}$. Note that this makes use of the old true penetrance estimates, the stategy-power exponent, and the elementary penetrance disparity in the unextended rectangles versus the extended ones.

The algorithm below adjusts the penetrance values according to this expression, and roughly estimates the error (in a manner similar to the approximating done in subsection 6.1.2).

```
COREXT($\theta$,G,h,hvar,S,S')

comment S = {$R_1$,$R_2$, ... ,$R_m$} is a subdivided set,
        S' = {$R_1'$,$R_2'$, ... ,$R_m'$} is the uncorrected extended
                                                        set of S;
Form immediate sets I = {$Q_1$,$Q_2$, ... ,$Q_m$}, from S,
                and I' = {$Q_1'$,$Q_2'$, ... ,$Q_m'$}, from S';
for j:=1 until m do;
    if $r(R_j') \neq r(R_j)$ then do;
        $val(R_j') := val(R_j) * [val(Q_j')/val(Q_j)]^{1/h}$;
        $dev(R_j') := devu(R_j) * val(Q_j')^{-\sqrt{hvar}/h} * val(Q_j)^{-\sqrt{hvar}/h}$;
comment {$R_1'$,$R_2'$, ... ,$R_m'$} is the output set;
```

The set **E** = COREXT(θ,G,h,hvar,S,S') is the (corrected) extended (region) set of **S'** (for θ, S, G, h, and hvar).

6.3.3. SUBDIVIDING THE EXTENSIONS

Let **E** be a corrected extended region set and **S** the unextended (subdivided) counterpart of **E**; let **G** be a final state graph set, and θ its associated evaluation function, with h and hvar having their usual meanings. Then extension subdivision proceeds:

```
                    EXTDIV(θ,G,h,hvar,S,E)

V  :=  {R' ∈ E | R' ⊅ R  ∀R ∈ S};
X  :=  SUBDIV(θ,G,h,hvar,V) ∪ (E - V) ;
```

X = EXTDIV(θ,**G**,h,hvar,**S**,**E**) is the _subdivided extended (region) set of_ **E** _(for θ, S, G, h, and hvar)_.

6.3.4. THE EXTENSION SUBSTEP

Suppose θ is an evaluation function, **G** is a final state graph set, h, hvar the stategy-power exponent and its estimated variance, and **S** is the subdivided set (of section 6.2). The full extension algorithm is given below.

EXTEND(θ,**G**,h,hvar,**S**)

```
S' := ENCLOS(S,G);
E := COREXT(θ,G,h,hvar,S,S');
X := EXTDIV(θ,G,h,hvar,S,E);
```

The first statement extends the rectangles, ignoring penetrance. The second corrects the penetrance values so that they remain estimates of the true ones. And the third statement subdivides the extensions.

The set **X** = EXTEND(θ,**G**,h,hvar,**S**) is the <u>final</u> (<u>region</u>) <u>set of</u> **S** (<u>for</u> <u>θ</u>, <u>**G**</u>, <u>h</u>, <u>and</u> <u>hvar</u>). If θ_{I-1} is the full penetrance function of iteration I-1, and $\mathbf{G}_I$, h_I, $hvar_I$, and $\mathbf{S}_I$ represent the usual variables, of iteration I, then $\mathbf{C}_I$ = EXTEND(θ_{I-1} , $\mathbf{G}_I$, h_I, $hvar_I$, $\mathbf{S}_I$) is the final or <u>established</u> or <u>cumulative</u> (<u>region</u>) <u>set</u> <u>of</u> <u>iteration</u> <u>I</u>.

<u>Example</u> <u>6.3</u>

The following is from iteration three of our standard example; it illustrates both extension and extension subdivision (i.e. the extension substep). The relevant region from the subdivided set was R = (r, 0.00001, 40.5), <u>lp</u>(r)=(11,0,0,0), <u>up</u>(r)=(21,2,1,0) and the corresponding uncorrected extended region was R' = (r', 0.00001, 40.5), <u>lp</u>(r')=(11,0,0,0), <u>up</u>(r')=(58,2,2,0). Note that extension occurred in the first and third dimensions.

The corresponding immediate regions were Q = (r, 86, 378, 1, 1) (from R) and Q' = (r', 129, 2377, 1, 1) (from R'). According to COREXT, the multiplier for the corrected extension is given by $k(R') = [val(Q')/val(Q)]^{1/h}$. The strategy-power exponent h was 0.34, so $k(R') = [(129/2377)/(86/378)]^{1/0.34} = (0.24)^{2.9} = 0.015$. Hence, after adjustment, $val(R') = (0.00001)(0.015) = 1.5 \times 10^{-7}$. This converts the strategy-dependent penetrance to an estimate of the true penetrance. (The deviation factors will not be calculated here.)

Subdivision also took place. Two regions resulted; the immediate subregions were $Q_1 = (r_1, 129, 457, 1, 1)$, $\underline{lp}(r_1)=(11,0,0,0)$, $\underline{up}(r_1)=(30,2,2,0)$ and $Q_2 = (r_2, 0, 1920, 1, 1)$, $\underline{lp}(r_2)=(31,0,0,0)$, $\underline{up}(r_2)=(58,2,2,0)$. $KFIX(\theta_2, h_3, hvar_3, R', \{Q_1,Q_2\})$ gives (intermediately, for i=1,2) $k(Q_j) = val(R')/avgval(\{Q_1,Q_2\}) = 1.5x10^{-7}/0.24 = 6.4x10^{-7}$. The bias for Q_1 is (0.66)(-0.72)(20.5-34.5) = 6.6. And the bias for Q_2 is (0.66)(0.72)(44.5-34.5) = -4.8. So the final values are $k(Q_1) = 6.4x10^{-7}*exp(6.6) = 4.6x10^{-4}$ and $k(Q_2) = 6.4x10^{-7}*exp(-4.8) = 5.3x10^{-9}$. And $val(Q_1)$ becomes $(129/457)(4.6x10^{-4}) = 1.3x10^{-4}$ while $val(Q_2)$ is $(0.5/1920)(5.3x10^{-9}) = 1.4x10^{-12}$.

6.4. THE ENTIRE PROCESS OF REVISION

In section 5.4 we saw precisely how the regions are created in the first iteration of a series. Now we can summarize the exact process of region revision for iterations after the first. This amounts just to the sequence of three substeps of the three sections preceding this one. If $\mathbf{C}_{I-1}$ is the cumulative region set of iteration I-1, θ_{I-1} and $\mathbf{G}_I$ the associated full penetrance function and final state graph set, then $\mathbf{C}_I$, the cumulative (final) region set of iteration I is calculated (for I > 1) by the plan **C** (see sections 4.3 and 5.4):

$$\mathbf{C}(\theta_{I-1}, \mathbf{C}_{I-1}, \mathbf{G}_I)$$

$\mathbf{U}_I$ = ULTREV $(\theta_{I-1}, \mathbf{C}_{I-1}, \mathbf{G}_I)$; (6.1.4)

$\mathbf{S}_I$ = SUBDIV $(\theta_{I-1}, \mathbf{G}_I, h_I, hvar_I, \mathbf{U}_I)$; (6.2.3)

$\mathbf{C}_I$ = EXTEND $(\theta_{I-1}, \mathbf{G}_I, h_I, hvar_I, \mathbf{S}_I)$; (6.3.4)

Summarizing, the first substep (with UTLREV) uses the true penetrance estimates u_0 from the old established regions, along with the corresponding elementary values u_s (count ratios); it assumes a relationship $u_s = u_0{}^h$, and revises the true penetrance estimates by averaging. This does not alter the rectangles. But the second step SUBDIVides the rectangles, again utilizing the new counts; it adjusts the penetrances appropriately (considering the elementary values to be biased). Finally, the third substep EXTENDs the regions from the second substep to cover outlying points, and refines the resulting rectangles.

If we were to focus attention on the region handling step we would notice that over a series of iterations, some regions become subdivided because of newly perceived differentiation in penetrance, and some remain intact, reflecting continuing uniformity of penetrance. The whole iterative process could be considered as an ongoing resolution of penetrance in the attribute space.

7. EXPERIMENTAL RESULTS AND SYSTEM PROPERTIES

The system was tested using the Honeywell 66/60 at the University of Waterloo. The first two steps were implemented in PL/1, and the regression step in Fortran. To reiterate, the user must supply the feature procedures (for the solving step). He also chooses three system parameters in the three respective steps: the cutoff M, the confidence factor c, and the confidence level $1-\alpha$. Their values can be altered by the user at each iteration if desired. In addition the user must select training sets.

This chapter describes the effect of the system when used with two different state-space problems, formula manipulation and the fifteen puzzle. The results are discussed in two respective sections.

Aside from the detailed results, some general tendencies can be detected -- i.e. there are properties exhibited by the plan within the context of the particular problem and feature set. This is not to say that the properties definitely generalize to all state-space problems, nor to all feature sets for a given problem, although there are frequently similar inclinations in the cases investigated (and although certain of the characteristics might be expected from the system design). Below are summarized some hypothesized properties of the system; the extent to which they may generally be valid is considered in the two sections to follow (where the discussion is interspersed with the detailed results).

P1. The splits are predominantly "correct" (i.e. the variation of the penetrance of regions versus their attribute representative is as would be expected).

P2. Although the modelled evaluation function does not necessarily reflect the splits precisely, it generally does. And from iteration to iteration, there is a loose correlation between the number and consistency of splits (in the region handling step), and the frequency of appearance of a feature in the model and its share of the sum of squares (in the regression step).

P3. Irrelevant features generally do not enter the model.

P4. A feature is rejected when subsumed by another, but not if the more general one is absent.

P5. The evaluation functions created from the regions have generally good performances.

P6. The functions improve in later iterations (i.e. chapter six is vindicated).

P7. The model is stable from iteration to iteration.

P8. The system is self-correcting.

P9. There are generally consistent results if an experiment is repeated with different training sets.

P10. There is convergence (i.e. the parameters stabilize).

P11. The convergence is to a local optimum of evaluation function performance.

P12. The plan is efficient.

The fifteen puzzle problem was chosen for its relative simplicity, so that the finer aspects of the system could be tested. In contrast, the intractable formula manipulation problem was selected to determine the limits of the system under unfavourable conditions. Let us begin with this latter problem.

7.1. FORMULA MANIPULATION

We have defined the fifteen puzzle and some features for it (chapter two), and to some extent we have also observed its character. Now we do the same for our other state-space problem.

7.1.1. THE PROBLEM

Our problem of formula manipulation is a restriction of theorem proving. With the apparatus described below it is possible to mechanize the proofs of some simple theorems which can be expressed in terms of a transformation of a (starting state) atomic formula (i.e. a formula without logical connectives), into a goal (atomic) formula. Shoenfield (1967) can be consulted for more general definitions than the ones we use.

A constant is a symbol with a fixed meaning. (We use the symbols 0, 1, a, b, and c for constants.) A variable can take on any constant value. (The symbols x, y, and z are reserved for variable names.)

An n-ary function f is a function mapping the n dimensional cartesion product C^n of the set of constants C into C. We define a term inductively:

(i) A constant, or a variable, is a term.

(ii) If $u_1, u_2, \ldots, u_n$ are terms, and if f is n-ary, then $fu_1u_2\ldots u_n$ is a term.

A <u>binary predicate</u> <u>P</u> is a subset of C x C (i.e. a binary relation). If u_1 and u_2 are terms, then Pu_1u_2 is a <u>formula</u>, which indicates that the pair (u_1,u_2) is in the relation P. (In our particular theorems, the only predicate that appears is the equality predicate.)

In the above definitions, terms and formulae are both in Polish prefix form; for clarity, we sometimes use the more familiar infix form, for example a=b in place of =ab.

In our formula manipulation problem, states are formulae.

We next require the operators of the problem. These are inference rules, combined with axioms. There are two rules of inference.

The first is <u>equality substitution</u>. Suppose that $\beta' = \delta'$ is an axiom and A is a formula which contains a term u, and that u and β' have a most general unifier σ, i.e. that $\beta = \beta'\cdot\sigma = u\cdot\sigma$.[1] Infer A' by replacing in $A\cdot\sigma$ some single occurrence of β with $\delta'\cdot\sigma$.[2]

The other inference rule is <u>modus ponens</u>. If $\beta' \longrightarrow \delta'$ is an axiom ("⟶" means "implies"), and A is a formula, and if β' and A have a most general unifier σ, then infer $\delta'\cdot\sigma$.

In the following subsection, examples are given of axioms and their use with these rules to become operators.

1 See the definition of unification in Nilsson (1971).

2 This is a simplification of paramodulation. See Robinson & Wos (1969).

7.1.2. PROBLEM INSTANCES

This subsection serves to illustrate the definitions of the previous one, and also to present the standard problem instance set used to measure the performance of evaluation functions. (As well, the method of generating training instances is given.) The set has five members, which are all very simple theorems from group theory. In the following, the theorem, its starting state, goal state, operators (axioms), and a solution for each are shown. (In addition, to be disregarded until the subsection to follow, the first three attributes of a seven attribute vector are listed for each state.)

T1: In a group, the identity is unique. Axioms used:

Ax1	=.xfx1	$xx^{-1} = 1$
Ax2	=.fxx1	$x^{-1}x = 1$
Ax3	=..xyz.x.yz	$(xy)z = x(yz)$
Ax4	=xy —> =.xz.yz	$x = y \longrightarrow xz = yz$
Ax5	=xy —> =.zx.zy	$x = y \longrightarrow zx = zy$
Ax6	.x1x	$x1 = x$
Ax7	=.1xx	$1x = x$

Solution (showing, in order, prefix form, infix form, axiom, first three attributes):

=.abb	$ab = b$		3	2	1
=..abx.bx	$(ab)x = bx$	Ax4	7	6	2
=.a.bx.bx	$a(bx) = bx$	Ax3	7	6	2
=.a1.bfb	$a1 = bb^{-1}$	Ax1	5	5	2
=.a11	$a1 = 1$	Ax1	2	2	1
=a1	$a = 1$	Ax6	0	0	0

The next two problem instances use exactly the same seven axioms.

T2: The inverse of the inverse of a is a.

=ffab	$(a^{-1})^{-1} = b$		3	2	2
=.1ffab	$1(a^{-1})^{-1} = b$	Ax7	4	4	3
=..xfxffab	$(xx^{-1})(a^{-1})^{-1} = b$	Ax1	8	7	3
=.x.fxffab	$x[x^{-1}(a^{-1})^{-1}] = b$	Ax3	8	7	4
=.a1b	$a1 = b$	Ax1	2	2	1
=ab	$a = b$	Ax6	0	0	0

T3: Cancellation law.

=.ab.ac	$ab = ac$		4	4	1
=.x.ab.x.ac	$x(ab) = x(ac)$	Ax5	8	8	2
=..xab.x.ac	$(xa)b = x(ac)$	Ax3	8	8	2
=.1b.fa.ac	$1b = a^{-1}(ac)$	Ax2	7	7	2
=b.fa.ac	$b = a^{-1}(ac)$	Ax7	5	5	2
=b..faac	$b = (a^{-1}a)c$	Ax3	5	5	3
=b.1c	$b = 1c$	Ax2	2	2	1
=bc	$b = c$	Ax7	0	0	0

T4: If any two idempotents of a group are permutable, their product is an idempotent. For this, axioms Ax1 - Ax7 were used, in addition to:

Ax8	=.ab.ba	$ab = ba$
Ax9	=.aaa	$aa = a$
Ax10	=.bbb	$bb = b$

Solution:

=..ab.abc	$(ab)(ab) = c$		5	4	1
=...ababc	$((ab)a)b = c$	Ax3	5	4	2
=..a.babc	$a(ba)b = c$	Ax3	6	4	2
=..a.abbc	$a(ab)b = c$	Ax8	5	4	2
=...aabbc	$((aa)b)b = c$	Ax3	6	4	2
=..aa.bbc	$(aa)(bb) = c$	Ax3	6	4	1
=.a.bbc	$a(bb) = c$	Ax9	4	2	1
=.abc	$ab = c$	Ax10	0	0	0

T5: If $\forall x(x^2=1)$, the group is commutative. The axioms used were Ax1 - Ax7, and:

Ax11	=.aa1	$aa = 1$
Ax12	=.bb1	$bb = 1$
Ax13	=..ab.ab1	$(ab)(ab) = 1$
Ax14	=..ba.ba1	$(ba)(ba) = 1$

Solution:

=.abc	ab = c		2	0	0
=.1.abc	1(ab) = c	Ax7	4	2	1
=...ba.ba.abc	[(ba)(ba)][ab] = c	Ax14	9	8	2
=..ba..ba.abc	[ba][(ba)(ab)] = c	Ax3	9	8	2
=..ba.b.a.abc	[ba][b(a(ab))] = c	Ax3	9	8	3
=..ba.b..aabc	[ba][b((aa)b)] = c	Ax3	9	8	3
=..ba.b.1bc	[ba][b(1b)] = c	Ax11	7	6	2
=..ba.bbc	(ba)(bb) = c	Ax7	5	4	1
=..ba1c	(ba)1 = c	Ax12	3	2	1
=.bac	ba = c	Ax6	0	0	0

These five problem instances might seem easy to solve using a graph traverser, but the state-space is very large, and if the search is breadth-first, the exponential growth rate is extreme. The equality substitution, especially, can result in large numbers of offspring, since substitution can be made for any term. With the axioms Ax1 - Ax7 (these prodide twelve basic operators, since "=" is cummutative), an estimated growth rate is very roughly $4^{d+d^2/2}$, where d is the search level.

The problem instances used for training were obtained as follows. First, the goal state:

(1) The first symbol of a partial formula G is "=".

(2) Randomly choose a symbol and append it to G. If G is not a substring of any well formed formula, go to (1). If G is a formula, stop,; it is the goal. Otherwise, go to (2).

From each goal formula G, a starting state was generated by repeatedly applying randomly selected axioms. This method allowed the creaticn of problem instances of any desired (maximum-length) shortest solution.

7.1.3. FEATURES

All our features $\{f_i \mid 1 \le i \le 7\}$ for formula manipulation measure differences between a given state A and the goal G. (They are similar to those of (e.g.) Popplestone (1967).) For each one except f_1, we define some function ξ and write $f_i(A,G)$ in terms of $\xi(A)$ and $\xi(G)$. (It is not necessary, but we define all the features so that they are "penalties", in order to simplify the relationships.) See the preceding subsection for numerical examples.

We begin with f_1, which tries to match terms of a state A with terms of the goal G, and totals the remaining uncorresponding symbols. To define it, we first need a simple concept relating two terms: $v \approx w$ if $v = w$ or if $v = fv_1 v_2 \ldots v_n$ and $w = fw_1 w_2 \ldots w_n$ and $\forall i$ ($v_i = w_i$ or one of v_i and w_i is a variable).

$f_1(A,G)$ (term matching)

```
comment G = $Pa_1a_2 \ldots a_\ell$ (where the $a_i$ are single symbols);
$f_1$ := length(A);
while there remains an unmarked symbol in A do;
    Find the longest unmarked term v in A;
    m := length(v);
    If $\exists$i so $v \approx a_i a_{i+1} \ldots a_{i+m-1}$, where no $a_{i+j}$ is marked then
        Mark $a_i$, $a_{i+1}$, ...., and $a_{i+m-1}$;
        $f_1$ := $f_1$ - m;
    Mark all the symbols of v;
```

The second feature compares the structure of a given formula A with that of the the goal G, ignoring differences in symbols. To define the feature f_2, we first need a recursive function which compares two terms or formulae:

```
                    structsc(B,C)

comment  B = $\beta_1\beta_2 \dots\dots \beta_n$,  C = $\delta_1\delta_2 \dots\dots \delta_n$;
If m = 0 then structsc := n
 else if n = 1 then structsc := 0
 else structsc := $\sum_{i=2}^{m}$ structsc($\beta_i$, $\delta_i$);
```

From this, we define $f_2(A,G)$ = length(A) - structsc(A,G) + length(G) - structsc(G,A).

The third feature compares depths of function nesting in the given state A and goal G. First we require a recursive function. If $B = \beta_1\beta_2 \dots\dots \beta_n$ is a term or a formula, then If n = 1 then depth(B) = 1 else depth(B) = 1 + $\max_i$[depth(β_i)]. And $f_3(A,G)$ = max[0, depth(A) - depth(G)].

The last four features are all defined in terms of a single primary binary function "count", which gives the number of symbols of a specified class in a formula B. Let S be the set of symbols. Count maps {B} x 2^S into the set of natural numbers, so that count(B,T) is the number of occurrences in B of a symbol of $T \subseteq S$. From count we define:

$f_4(A,G)$ = max[0, count(A, {.}) - count(G, {.})].

$f_5(A,G)$ = max[0, count(A, {f}) - count(G, {f})].

$f_6(A,G) = \max[0, \text{count}(A,\{a,b,c\}) - \text{count}(G,\{a,b,c\})]$.

$f_7(A,G) = \max[0, \text{count}(A,\{1\}) - \text{count}(G,\{1\})]$.

7.1.4. RESULTS

The evaluation functions created by the plan for this feature set are not very high performance ones.

One difficulty is that the fast node growth of this formula manipulation problem inhibits system functioning. According to the rough estimate of breadth-first node growth, the number generated at level d is $4^{d+d^2/2}$, so for d = 0, 1, 2, 3, ..., the values are 1, 8, 260, 32000, ... This means that the first iteration, with its trivial evaluation function, cannot solve problem instances whose shortest solutions have a length of more than two or three. (And in post-initial iterations, the situation is not much better.) Hence the good count (number of solution nodes) is generally low. Furthermore, because the growth rate is so high, most nodes generated are not developed, so the total count is also low. In turn, the small values of the counts cause little splitting (see the definition of dist in chapter five), so not much discrimination is gained in a single iteration (i.e. the evaluation function cannot improve much).

Because of the nature of this problem, two system parameters were set so as to effect faster results. The one, c, was given a value of 1.0. (Recall from chapter five that c influences the likelihocd of splitting -- smaller values are equivalent to lower confidence levels.) This means that splitting occurs if the "sureness" of a penetrance difference is only about 70%. The other system parameter, a, was set to 0.25. (Recall from chapter four that $1 - a$ is the confidence level for entering features into the modelled penetrance function.)

The results of one iteration series are summarized in tables 1.7.1 to 1.7.3, which give the relevant values for the solving step, region handling step, and regression step, respectively.

Table 7.1.1 shows the results for the solving step over three iterations. The second column gives the maximum shortest solution lengths of the problem instances presented (this training set having been generated as described in subsection 7.1.2). The third column lists the number in this set, and the number of those which were solved. The fourth column shows the average number of nodes developed before a solution was found. This column also gives the average number of nodes generated. In every case, the cutoff (maximum allowed generated before abandoning the search) was 5700, and in instances for which no solution was found, the contribution to the average could only be known to be larger than 5700, so the corresponding entries reflect this. The final column shows the average solution length

Table 7.1.1. Solving Step.

Iter-ation	Max Depth	Number Solved, Total	Avg Nodes Developed, Generated	Average Path Length
1	2, 3	9, 9	33, 632	2.2
2	3, 4	10, 15	>63, >2700	3.2
3	3- 6	8, 11	>30, >3300	4.5

Table 7.1.2. Region Handling Step.

Iter-ation	Feature Number						
	1	2	3	4	5	6	7
1					1,		1,
2	3,1	3,		1,		1,	
3							

Table 7.1.3. Regression Step.

Iter-ation	Parameter Number								Er-ror
	0	1	2	3	4	5	6	7	
1									
2	0.22	-0.16	-0.22	-1.05			-2.26	-2.62	0.11
3	-0.06	-0.19	-0.23	-1.27				-2.01	0.12

(approximate when not all instances were solved). This column suggests a slight progressive improvement in evaluation function performance.

Table 7.1.2 lists the splits in each iteration. The entries are pairs; the first number of the pair gives the splits that occurred in the expected manner (i.e. reflecting the expected penetrance versus attribute variation), and the second integer gives the number that occurred in the contrary direction. A blank indicates no split. The table shows that three regions existed at the end of the first iteration, and twelve at the end of the second. In the third, there was no further differentiation.

Table 7.1.3 shows the parameters b_i for the modelled penetrance function $\rho = \exp[\underline{f}.\underline{b}]$. (The final column gives the residual error for the log-linear model.) No regression was performed in the initial iteration (so the second row of entries in table 7.1.1 are for the grouped function ν_1 only). In the second iteration, all features but f_4 and f_5 entered the model. Iteration three parameters were similar (except the accepted features were f_1, f_2, f_3, and f_7), since the regions had been altered only by having their penetrance values revised, not the rectangles.

The performances of the evaluation functions are indicated in table 7.1.4. The triple entries give number of nodes developed, number generated, and solution length (if any found before cutoff), respectively. Since early experiment showed little difference in general performance in later iterations, whether the full evaluation function θ or just the modelled component ρ was used, the simplest choice was to make measurements for ρ only. The table gives the performance with the five standard group theory instances of subsection 7.1.2 (any solutions were similar to those in that subsection). The functions tested were θ_0, ν_1, ρ_2, ρ_3, and also two others, ρ_{3a} and ρ_{3b}. These latter two were obtained from ρ_3 by slightly disturbing two or three parameters b_i of ρ_3. The function ρ_{3a} is identical to ρ_3

Table 7.1.4. Evaluation Function Performance

	Theorem				
fn	T1	T2	T3	T4	T5
θ_0	>5700	>5700	>5700	>5700	>5700
ν_1	>5700	>5700	>5700	>5700	>5700
ρ_2	22,314,5	85,2342,7	79,2564,5	26,364,8	>5700
ρ_3	21,302,5	98,2714,7	43,1046,7	26,364,8	>5700
ρ_{3a}	55,1266,5	>5700	13,272,7	26,364,8	>5700
ρ_{3b}	19,274,5	192,5289,7	23,366,7	26,364,8	>5700

except $b_5 = b_6 = -2.0$, and ρ_{3b} is the same as ρ_3, except $b_5 = b_6 = -1.1$, and $b_3 = -0.75$.

The functions ρ_2 and ρ_3 show a definite improvement in performance (if $d = 7$, $4^{d+d^2/2} \doteq 10^{19}$).

Another experiment resulted in ten regions before splitting stopped. Although the other parameters were similar to the ones above, those for f_4 and f_5 were positive, which gave a poor evaluation function. This cccurred largely because the system parameters c and $1-\alpha$ were set so low, and suggests that interactive guidance is required when the system is pushed to the limit.

The comparative performance of our graph traverser with ρ_3 is not impressive; for example, Popplestone (1967), using similar features, obtained significantly better results, albeit with a more sophisticated traverser (using partial development). (E.g. T5 was solved, and with only a few tens of developed nodes.) See also Huet (1971).

Despite the intractability of the problem, there is evidence for certain of the properties hypothesized at the beginning of this chapter. The splits are predominantly as expected (P1). The evaluation function performance is perhaps reasonable considering the simplicity of the graph traverser and difficulty in obtaining information for penetrance differentiation (P5). And the final functicn ρ_3 was constructed by solving 35 problem instances (giving only 88 solution nodes in total) (P12).

7.2. THE FIFTEEN PUZZLE

This section describes substantial results obtained with the fifteen puzzle, which is more manageable than the problem of the last section. Although there are about ten trillion states in the state-space, the exponential breadth-first growth rate is only about 2^d (where d is the level). Moreover, in some intuitive sense, this puzzle seems relatively simple, having considerable "structure".

There are four subsections. The first describes details, particularly of two main experiments, and at the same time, some properties of the system are further examined. The second two consider the computational cost of the main results, especially in comparison with more direct methods of parameter tuning. Finally, the fourth subsection relates the present research to some other work.

7.2.1. EXPERIMENTAL RESULTS

As with the previous problem, puzzle instances were obtained in two ways. The training set was generated as follows. The goal (illustrated in chapter one) was progressively transformed toward an ultimate starting state. The randomly selected successive operator applications were terminated when the desired path length d was reached. Hence d became the upper limit on the shortest solution for the resulting problem instance.

The other generation method supplied instances for measuring the performance of evaluation functions. Each "average" puzzle instance of unknown difficulty was created by randomly selecting hexadecimal digits, and appending them to a growing string (barring earlier occurrence), continuing until it reached the proper length of sixteen. (And of course odd permutations were rejected, since they are unsolvable.) A standard set of such random puzzles was produced.

There are two major experiments reported here. (An abbreviated version appeared in Rendell, 1977.)

The three tables below summarize the results for experiment one, a series of seven iterations which used all six features of subsection 2.2.2.

Table 7.2.1. Series One Solving Step

Iter-ation	Max Depth	Number Solved, Total	Average Developed	Average Path Length
1	8,9	6, 6	580	8.3
2	15	7, 14	>>931	~18
3	75	12, 15	>532	~88
4	75	9, 12	>705	~96
5	75	11, 12	>462	~95
6	100	9, 12	>717	~98
7	100	8, 12	>740	~111

Table 7.2.1 lists information about the particular set of puzzles for each solving step. The second column gives an upper limit d for the solution length of the training puzzles. At the outset, when the search is breadth-first, d can be about nine at most, if the puzzle is to be solved. Except for the first iteration, when the cutoff was 2200, a maximum of 1500 nodes was allowed. The third column lists both the number of instances in the training set, and also the number solved before the graph traverser abandoned the search. The fourth column gives the average number of nodes developed before a solution was found. (The number generated was typically slightly greater than twice the number developed.) When the traverser failed to solve a problem, only a minimum value could be known; accordingly the average number developed has a lower limit (reflected in the table by '>'). Similarly, the entries in the fifth column, which indicate the average solution length, are approximate where indicated.

Between a couple of hundred and a thousand different attribute space points resulted from the solving step (increasing in later iterations).

Table 7.2.2. Series One Region Handling Step

Iter-ation	Feature Number						To-tal
	1	2	3	4	5	6	
1	2,						2
2	3,	1,				2,	6
3	2,		2,		1,1	,1	7
4	1,	1,			2,1		5
5		1,		1,	,1	3,2	8
6	1,		1,	5,	1,		8
7	1,	2,2			5,	,1	11
Total	10,	5,2	3,	6,	9,3	5,4	47

Table 7.2.2 summarizes the results of the region handling step. Most of the splits which occurred were "expected" and a few were "unexpected", or contrary to the general trend. These two categories are listed separately, separated by commas. Below each feature number, the "expected" splits are to the left, and the "unexpected" to the right. A blank indicates zero.

The confidence factor c was 1.5 throughout.

Table 7.2.3. Series One Regression Step.

Iteration	Parameter Number 0	1	2	3	4	5	6	Error
1	(1.46)	(-0.88)						(0.07)
2	4.34	-0.70	-1.92		-2.24			0.13
3	2.16	-0.79		-2.61				1.17
4	0.92	-0.57	-0.84	-2.43				2.30
5	-1.46	-0.52	-0.65					3.12
6	-0.90	-0.49		-1.77	-0.52			1.96
7	-1.03	-0.41	-0.83	-1.67	-0.47			1.65

Table 7.2.3 lists the parameters b_i computed by the regression step for the log-linear modelled penetrance function $\rho = \exp[\underline{f}.\underline{b}]$. Blanks denote zeroes, the stepwise regression procedure having rejected the corresponding features. The entries are bracketed where the number of regions had not yet reached the number of features (and the model was in fact not used). The "error" column lists the logarithmic residual regression errors. For each iteration, a confidence level of 0.85 was used.

Comparing tables 7.2.2 and 7.2.3, we can see that the non-zero parameters calculated by the system do not necessarily conform strictly to the splits of the region handling step (of the same iteration). In fact, a feature can enter the model despite no split having occurred in the corresponding dimension. For example, in iteration two, there are two splits in the sixth dimension, but the sixth parameter is zero. In contrast, there has been no split in the fourth dimension at this point, but the fourth parameter is non-zero (this effect is caused by SHRINK). However, there is a general strong agreement between the number and consistency of splits for a feature in the region handling step, on the one hand, and the likelihood of its being in the model, on the other. For example f_1 has a total of ten splits overall, all of which reflect "expected" penetrance variation (table 7.2.2 clearly shows f_1 to be the most prominent feature), and it is also the only feature never to leave the model. (In addition, it is consistently the feature that absorbs the largest part of the regression sum of squares.) At the other extreme, one might easily predict from table 7.2.2 that f_6 is not likely to be prominent in the model (and in fact it never enters). This discussion more or less restates properties P1 and P2 (moreover it is the general pattern exhibited in other experiments).

It is apparent from table 7.2.1 that the evaluation function improves, but before we examine its performance in detail, let us consider another series of iterations.

The second experiment required fewer iterations before convergence. It used just the first four of the above features, the ones that appeared in the final model of that series. The training set consisted of entirely different puzzle instances. The results are summarized in tables 7.2.4 to 7.2.6, which are analogous to tables 7.2.1 to 7.2.3. (As before, the confidence factor c (for region splitting) was 1.5, and the confidence level (for regression) was 0.85.)

Table 7.2.4. Series Two Solving Step

Iter-ation	Max Depth	Number Solved, Total	Average Developed	Average Path Length
1	9	4, 4	713	9
2	12-14	12, 12	248	14
3	50-75	8, 12	>637	~84
4	100	7, 12	>660	~104
5	100	12, 12	373	104
6	100	13, 14	>367	~102

Table 7.2.5. Series Two Region Handling Step

Iteration	Feature Number				Total
	1	2	3	4	
1	2,				2
2	2,	1,			3
3	1,		1, 1	3, 1	7
4	1,	2,			3
5				, 1	1
6					0
Total	6,	3,	1, 1	3, 2	16

Table 7.2.6. Series Two Regression Step.

Iteration	Parameter Number					Error
	0	1	2	3	4	
1	(1.01)	(-0.81)				(0.02)
2	0.73	-0.72	-2.42			0.21
3	0.25	-0.51	-2.15	-1.62		1.46
4	0.70	-0.56	-2.18		-0.67	1.22
5	0.59	-0.52	-2.17	-1.14	-0.52	1.08
6	0.45	-0.49	-2.27	-1.34	-0.53	0.89

As might be expected, it seems that more iterations are required when there are more features (experiment one needed seven iterations, whereas in series two the model was nearly static after the fourth iteration).

We shall shortly look into the effectiveness of the evaluation functions generated in these experiments, but first let us examine some additional properties exhibited by the system, at least in the context of these two examples. These two series of iterations (and others) show the general stability of the system. The parameter values tend to change just gradually from one iteration to the next (P7). Tables 7.2.2 and 7.2.5 seem to indicate that the region handling step does succeed in revising penetrances toward their "correct" values, since the regression error decreases whenever splitting does not predominate. This correction amounts to a stabilizing negative feedback.

There is a related tendency: Often, when a parameter is zero for an "important" feature, an iteration causes splits in that dimension, and the parameter becomes non-zero (e.g. series one, iteration 6). This is what one would expect. (In fact the plan corrects itself to a greater degree than this. In an entirely different experiment, the two system parameters c and a were 1.0 and 0.25, respectively, and positive parameters arose. This resulted in a temporarily poorer evaluation function, but the bias term in the distance function now caused many "correct" splits in the corresponding dimensions (see the full definition of dist in chapter six). This, in turn, meant an immediate correction in the regression step -- evidence for P8.)

There is also a consistency over different iteration series (P9). Series two gives results very similar to those of the first experiment. (Compare tables 7.2.3 and 7.2.6 -- some corresponding parameters are identical.) Furthermore, the models converge (P10).

Table 7.2.7. Performance of Functions

	Series One			Series Two		
Iter-ation	Percentage Solved	Nodes Devel.	Path Length	Percentage Solved	Nodes Devel.	Path Length
2	97	>550	~125	77	>703	~116
3	68	>831	~120	94	>614	~118
4	94	>568	~126	100	427	116
5	67	>770	~122	100	371	118
6	94	>570	~141	100	371	118
7	100	353	113	-	-	-

Table 7.2.7 indicates the performance of the modelled evaluation functions generated in both series. A sample of

thirty-two completely random puzzle instances was used. The cutoff (upper limit of generated nodes) was 2200 (i.e. a maximum of about 1100 could be developed). As in tables 7.2.1 and 7.2.4, any unsolved puzzles count as having greater than 1100 nodes developed, so the average values are sometimes approximate.

From table 7.2.7, it can be seen that the performance improves very quickly at the beginning, then more slowly. In series two, the modelled penetrance function provided by iteration 2 already solved three quarters of the completely random puzzles, while the evaluation function of series one iteration 2 solved nearly all of the standard set. However, excellent performance was not attained until iteration 7 (series one).

Let us examine in more detail the statistics obtained from the performance measurements of this best penetrance function, which we can denote by ρ^*. Another eighteen random puzzles were added to the sample, so ρ^* was tested with a total of fifty. All were solved; the longest solution was 178 and the maximum number of states developed for any instance was 975. The average number developed before solution was 353.5, and the sample deviation was 180. (The average effective branching factor was very small, less than 1.005.) Let us assume that the distribution of the number developed before solution is log-normal. From the numerical values it was determined that the estimated logarithmic mean and standard deviation were 5.77 and 0.48. If these are taken as the true logarithmic mean and standard deviation, it follows that ninety-nine out of one hundred random puzzles will be solved by developing no more than 970 nodes.

The system has calculated parameters which are locally optimal (series one) or else very close to it (series two) (P11). To test this, the parameters of ρ^* (iteration 7 of example one) were varied one at a time (actually those for f_2 and f_3 were varied together; f_2 and f_3 are similar) and the resulting linear polynomial was used as an evaluation function with the standard set of thirty-two random puzzles (again with a cutoff of 2200 -- or about half that number developed). The average solution length and number of developed nodes are graphed in figures 7.1 and 7.2. Both the path length and number of nodes devoloped are plotted on each graph; the scale on the left refers to average number of developed nodes, and that on the right, to average solution length. Dots with circles represent nodes developed. (Open symbols indicate the lcwer limits in cases where not all puzzles were solved). And crosses represent solution lengths. (Typical estimates of the standard deviations were about two hundred for developed nodes and about twenty-five for path lengths).

Although series two did not produce parameters which are identical to those of series one, the only discrepancy arises on the flat portion of the f_2, f_3 curve, and this is

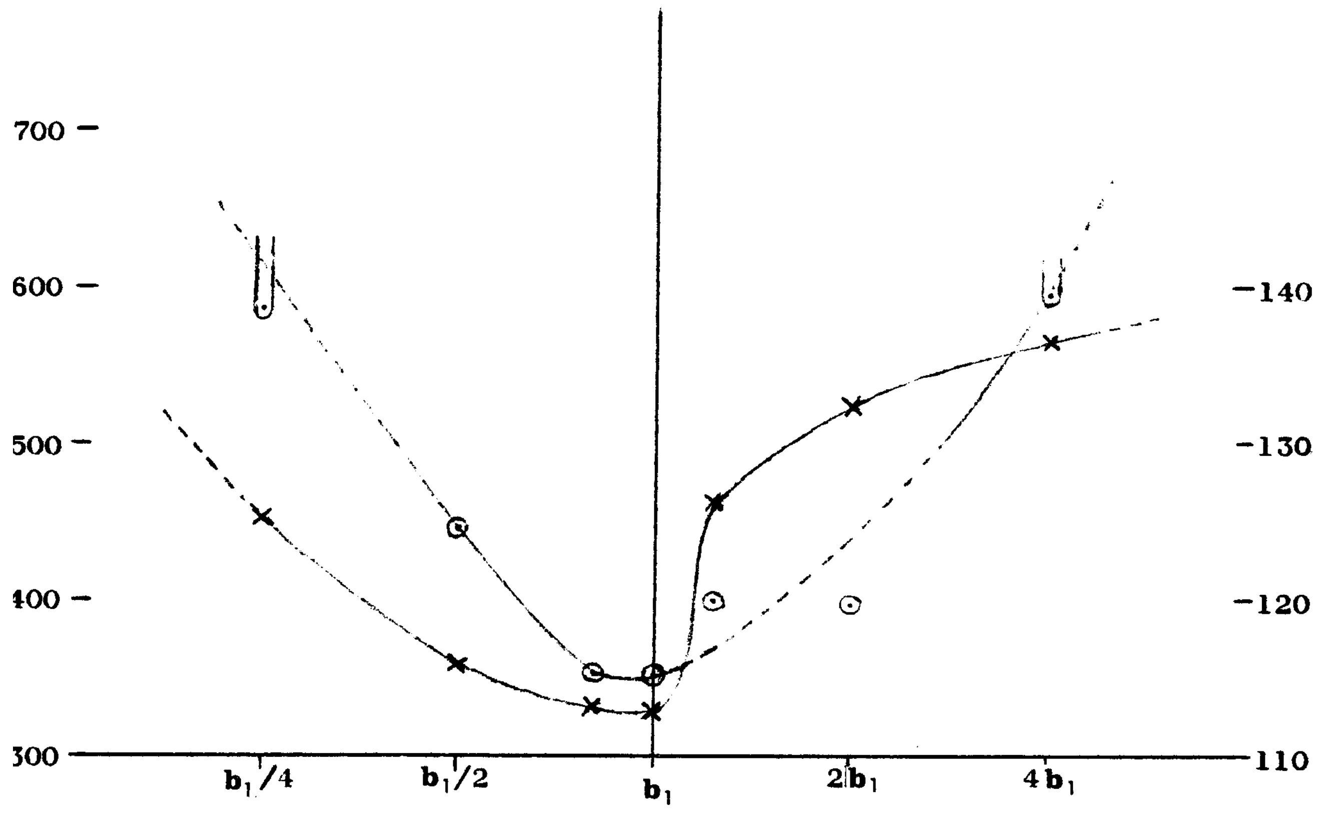
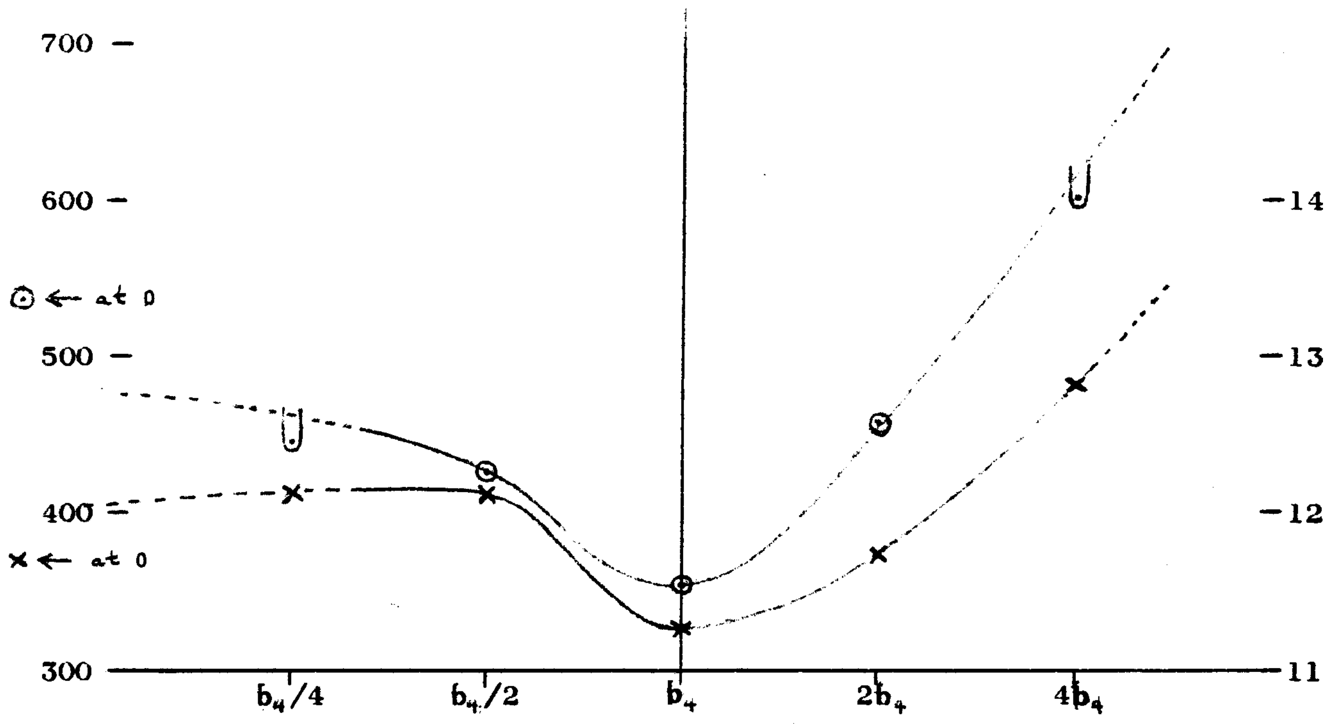


Figure 7.1. Performance of system-generated evaluation function ρ^* (series one iteration 7), with disturbed values of the first parameter b_1. The graphs show the average nodes developed (circled points and scale on left) and average solution lengths (crosses and scale on right) vs b_1 for the standard random set of 32 puzzles. The center line represents the value computed by the system.



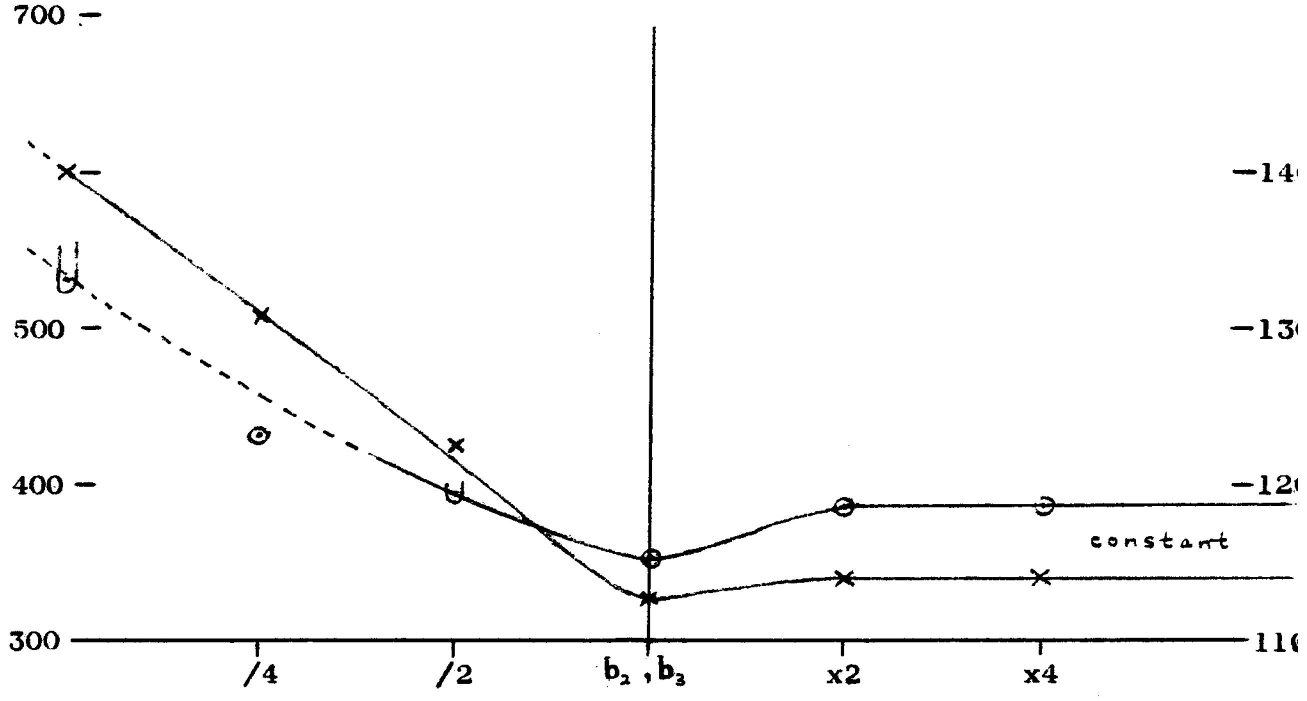


Figure 7.2. Average nodes developed and path lengths vs b_4 (upper) and b_2, b_3 (lower graph).

only a little above the minimum.

It is interesting that the locally optimal parameters are in a nearly integer propcrtion.

A few program runs were made using the best modelled function ρ^*, but adding slightly negative parameters for f_5 or f_6. No improvement could be found.

In experiment one, there is evidence for hypcthesized properties P3 and P4. The fifth feature f_5 is subsumed by f_2 (see subsection 2.2.2). And f_5 was rejected. When a different experiment was conducted, using just f_1 and f_5, f_5 of course played an important part. (Again a local optimum was found; the parameters b_1 and b_5 were in the approximate ratio 1:4; and 69% of the standard random puzzle set were solved, with an average number of developed nodes of >811 and average solution length of 122.)

For the final iteration of experiment one (which gave ρ^*), several plots were made of the residuals. Although the patterns were not strong, the results indicated that an f_1^2 term might be beneficial, and that there seem to be some possible interactions. We return briefly to these subjects of residuals and nonlinearity in chapter eight.

7.2.2. COSTS

This subsection considers the computational cost of the fifteen puzzle experiments. In addition to the those already reported, various test runs were made to estimate the complexity introduced by elements of the system.

The seven iteraticns of experiment one took a total of 63.6 minutes on the Faculty of Mathematics Honeywell 66/60 computer at the University of Waterloo. Ninety-one percent of this was required by the solving (and point-generating) step (step one -- the graph traverser). The other nine percent was taken by the region handling step (step two). (The regression step needed an insignificant time -- less than one half of one percent of the whole.) Step one used an average of 91k words, and step two 59k.

The six iterations of experiment two required 25.6 minutes, with only five percent of this (1.3 minutes) being used for region handling. In contrast, 4.2 times this amount was used for region handling in iteration one (or 3.6 times, per iteration). According to subsection 5.2.2, the speed of CLUSTER decreases with the square of the number of features; here this factor is $(6/4)^2 = 2.2$. The discrepancy is caused by the facts that in experiment one, both the number of regions and the number of attribute points are generally larger (both CLUSTER and SHRINK are called once for each established region).

In an attempt to reduce costs, experiment two (four features) was repeated twice but with more permissive values for the confidence factor c (region handling step) and confidence level $1-\alpha$ (regression step). The values were c=1.3 and $1-\alpha = 0.55$. Also, for both these experiments (2a and 2b), fewer puzzle instances were used in training -- six for each iteration except the first two (which had one, two or four).

After seven iterations, experiment 2a gave the parameter vector (const, -0.38, -0.90, -0.46, -0.56). Using the same random test set as before (32 instances), the evaluation function solved 100%, with an average number of developed nodes of 355, and an average solution length of 113. Experiment 2a required 13.6 minutes in total.

For experiment 2b, further economy was obtained by lowering the node development cuttoff (to 700). Here, however, eight iterations were needed before convergence. The resulting vector was (const, -0.64, -1.18, -0.62, -0.78). Its performance was 100% solved, average nodes developed 356, and average solution length 118. The total time used was 9.5 minutes.

Among the factors influencing the graph traverser speed is one aspect of the data structure for nodes. After their generation, new states need to be compared to old ones for a redundancy check (the former are dismissed if they already exist). An ordered binary tree is implemented. From experiment it was determined that over the cutoff range ever used (for the fifteen puzzle) the speed was almost independent of the tree size. About 25 nodes per second were generated and retained (with four features when there was no attribute point generation).

Running step one takes slightly more resources if attribute points are generated. In order to save space and to speed later clustering, counts are used rather than multiple point instances, and the linear search implemented to locate points is slow. With this scheme, these data typically required about 10%-15% of the total storage. And, typically, their generation also needed 15%-20% more time.

(Our) features cost little, only about 7% extra time for each (relative to the case without point generation). Consequently, there seems to be little question that increasing the number of features (for at least the ones we have been considering) can be cost-effective. For example, comparing ρ^* with the optimal distance score and reversal score function ρ^+ (100% vs 69% puzzle instances solved, 353 vs >811 average nodes developed, 113 vs ~122 average solution length), we have: The latter developed 130% more nodes. Also, the quality of the ρ^+ solutions was worse, by 8%. But the marginal cost of two extra features is only about 14%. In this case, at least, "expert knowledge buys expert performance".[1]

[1] See also the discussions in Gaschnig (1979).

7.2.3. LESS MECHANIZED ALTERNATIVES

Let us now compare our main fifteen puzzle results with more direct methods. The easiest is simply to vary the parameters systematically, beginning with some reasonable guesses. But this would obviously be inferior; perhaps thousands of test instances would need to be solved.

However, there is the possibility of probit (or logit) analysis,[1] a technique which can avoid regions and directly fit individual points to discover a relationship between the penetrance u and the attribute vector $\underline{x}$. Proportions such as our penetrance (or their difference from unity) usually have an "S" shaped distribution (steadily rising, assymtotic to one), and this curve can be formalized as (e.g.) the cumulative normal distribution. The u-transformation that will linearize this function is the probit. Our individual penetrance observations are binary, either zero or one, depending on whether the corresponding node was in a solution. The methcdology appropriate for fitting binary data is called maximum likelihood, which uses an iterative process. In the case of the probit transformation, a parameter vector $\underline{b}$ is computed[2] which will maximize the likelihood of the cumulative ncrmal value of $\underline{x}'.\underline{b}$ predicting

1 See Bartlett (1947) and Finney (1952).

2 A program which incorporates the probit transformation and maximum likelihood was obtained from Dr. R. Shillington, Policy Research and Management Services, Saskatchewan Health, Regina.

the observed u's.

In experiments with this direct method, the overall procedure was similar to the operation of the system, but without the region handling step of the latter: First a training set of problem instances was solved, then the resulting data were analysed to compute parameters for the succeeding iteration. (Both best subset feature selection and forcing were used.) Note that the data points were utilized directly, without conversion to estimated true penetrance. Also, there was no accumulation of penetrance information from iteration to iteration.

Features f_1 through f_4 were chosen. The conditions were matched to those for the system experiments -- easy puzzles were chosen for the first iteration, then harder ones as their solution became more likely. For each iteration a few thousand data points resulted (one for each training graph node). Perfcrmance measurements were also similar; a random set of 32 puzzles was used, and average numbers of developed nodes were compared.[1]

These experiments involved about forty program executions in all, varying the numbers (4 to 30) and difficulty of training problem instances (shortest solution 12 to 28). Generally the consequent evaluation functions were good but none performed as well as those found by the system. The

[1] This was the same standard random test set used in earlier performance measurements.

magnitudes of paramters b_2 to b_4 were consistently lower than optimal. This was because of an inherent limitation of the approach, described in the following typical outcome.

The only relevant feature in the first iteration was f_1 (distance score). Its parameter b_1 was -0.52.

Eleven puzzle instances with maximum shortest solution length between 24 and 28 constituted the training set for the second iteration; eight of these were solved. Now the other three features entered the model. The new parameters were $b_2 = -0.56$, $b_3 = -0.51$, and $b_4 = -0.39$. However, b_1 became -0.004, much smaller than in iteration one. This of course was because the first iteration (using a breadth-first search) reflected the true penetrance more accurately, whereas the second had points biased by the non-zero parameter -0.52. The resulting second iteration parameter vector is almost equivalent to one which has b_1 any negative finite quantity, and the other parameters all minus infinity. This vector resulted in an average number of developed nodes of >679, with 81% of the standard random test set being solved.

A more sensible vector can be obtained by selecting the unbiased value for b_1 from the first iteration, and the other (unbiased) parameters from the second.[1] This gives a vector quite close to the (locally) optimal one found by the

[1] These parameters are unbiased except for the fact that they were not obtained from breadth-first searches for <u>random</u> puzzles.

system, and the performance of this vector was good: The average number of nodes developed was >436, with 94% of the puzzles solved (compared with 353 and 100% for ρ^*).

Next, a third iteration was attempted, using the appropriately mixed parameter vector from the first two iterations: (const, -0.52, -0.56, -0.51, -0.39). This time a more serious problem arose. The parameters actually became positive (const, 0.04, 0.09, 0.14, 0.09).[1] Without some scheme for blending these data with former penetrance information, they are completely useless. (In contrast, the system provides a means for gradual adjustment of cumulative information, with feedback seemingly correcting "wrong" modifications.)

But the costs of the two successful iterations of the above process were as follows: The two solving steps took 8.5 minutes. The maximum liklihood routine, operating with the user forcing exactly the desired variables (this being the most advantageous), took a total of 3.1 minutes. And the entire procedure was 11.5 minutes.

Summarizing, this direct method may be useful for introduction of new features, but not for refining parameters. It might be expected to give fairly good results when there is little or no interaction amongst features, and when the non-randomness of early training problem instances is not too important.

[1] Attempts to incorporate feature interactions gave rise to similar incapacitating effects.

Since the parameter vector discovered in this way is close to (locally) optimal, a traditional method of optimization could now be much more effective. To test this, a number of experiments were conducted which fitted the four parameters b_1 through b_4 to a performance response surface (here the parameters b_i have become independent variables). The minimum was located by differentiating. An observation was obtained by selecting the vector (b_1, b_2, b_3, b_4) then solving a (usually) randcm puzzle instance to obtain the number cf nodes y developed in the search. Many parameters of this process were varied, including the number of points fitted, the types of terms in the response surface model, and transformations both of the b_i's and of the y's. The procedure worked well only if the number of points was quite large; otherwise either the fitted surface was not convex or else the resulting evaluation function was no better than the starting one. The sequel describes the details.

There is often difficulty in choosing values of the independent variables for respcnse curve fitting. If they are too close together, the random errors tend to cverride the underlying relationships. If they are too far apart, the real surface may not match the hypothesized one. In our case the points cannot be too far apart also because bad parameter vectors cause the graph traverser to fail too often (alternatively, the cost of a large increase in the cutoff is prohibitive). But we already have some information from the graphs in figures 7.1 and 7.2 (earlier in this chapter). These graphs suggest that if the coordinates of a point close to the optimum are varied by a factor of 2 or so (perhaps 4 for b_1), the surface has a shape well described by a model with second order terms. If the points are much farther apart than this, the response y is indeterminate in practice. So the points were selected accordingly: They were the corner and all midpoints of subsurfaces of the hypercube with center (-0.52, -0.56, -0.51, -0.39) and side d in the logarithmic four dimentional space. The values of d were 1.8 and 2. There are 81 such points (for a given d), but 16 of them are actually redundant because multiplication by a constant factor leaves the vector essentially unchanged in our case. This leaves 65. In the experiments, sets of 65 points provided the data; and two groups (with different d's) were sometimes used. (Occasionally a few other haphazardly selected points were added.)

We have additional knowledge about the character of this situation: There is probably a discontinuity in the surface where any b_i is zero. Thus a logarithmic transformation may be beneficial (and, again, the graphs suggest a logarithmic transformation anyway, as does the fact that it is the <u>relative</u> values of parameters that matter). In addition, a logarithmic transformation of y is also indicated, since zero is its lower bound. Experiments supported both of these conclusions; logarithmic transformations almost always improved results (there were fewer instances of useless fits, and consequent evaluation functions performed better). So $\log|b_i|$ and $\log(y)$ became the standard variables.

For number of points n even as large as 145, it was found that the 15 term model with cross terms never worked well (the 8 term model without interactions was always superior). This seems reasonable in view of the large deviations and the shapes of the graphs in figures 7.1 and 7.2. Thus attention focused on the 8 term model.

For several different runs with d=1.8 or d=2, and n=40, 65 or 80, no improvement was found over the original vector (-0.52, -0.56, -0.51, -0.39).[1] It was clear, however, that larger values of n improved results. When n was increased to 145 (d=1.8 and d=2), a better vector was found: (-0.43, -0.72, -0.63, -0.50). The evaluation function solved all but one of the 32 in the standard test set (97%), with average number of developed nodes >366 and average solution length ~116.

The cost of the 145 points was 45.1 minutes (the cost of the fit and location of the minimum was negligible -- 0.05 minutes). So the two stage method (first probit

[1] Different random puzzle instances were used for each point, except one experiment instead used a single fixed puzzle instance for each point (n was 80). As might be expected, the residuals were small but the minimum was perverted.

analysis, then optimization) took a total of 56.7 minutes.

In conclusion, the more elegant and automatic system seems to be superior to the two stage method of probit analysis and optimization in terms of some combined measure of reliability and efficiency. Two possible reasons for this are: (1) the cumulative (and natural) regions permit stability through gradual penetrance adjustment, and (2) attribute space localization of the performance measure may result in a greater quantity of meaningful information per problem instance; whereas the traditional optimization method yields one quantity per search (the total number of nodes developed), the plan yields as many measurements as there are regions. Furthermore, the plan can probably be improved; some ideas are summarized in the next and last chapter.

7.2.4. COMPARISONS WITH OTHER RESEARCH

This subsection considers the main product ρ^* (which solved all puzzle instances presented and had locally optimal parameters) in the light of other work. Also some of the characteristics of ρ^* are examined.

There have been no reports of other one-way graph traversers having such a degree of success with the fifteen puzzle. In an early system of Doran & Michie (1966), the experimenters themselves varied the parameters (of a model somewhat different from ours -- see section 1.2). With the best choice, the program succeeded in solving six out of ten of the random puzzles they tested, although that node generation cutoff was lower (500). When Doran and Michie's evaluation function was used with a cutoff of 2200, 84% of our standard set were solved, and an average of >550 nodes were developed, with an average solution length of ~138.

Of course the choice of features is of central importance (ours was an advantageous one, selected by the human experimenter).

A two-way traverser has solved the fifteen puzzle (Chandra, 1972), with about the same number of developed nodes. But, as mentioned in subsection 2.1.2, that program had several advantages. Aside from the increased power of the two-way traverser,[1] the program included a specific type of problem reduction, designed especially for the fifteen puzzle. The present system is more general. Although features are defined and selected by the user, and although they embody specialized knowledge about a problem, the plan screens and weights them.

Let us now consider our function ρ^* from a different perspective. Section 1.2 briefly summarized some of the work of Pohl (1970) and of Gaschnig (1979), who schematized the evaluation function as $f = (1-w)g + wh$ $(0 \leq w \leq 1)$. We

[1] See Pohl (1969) for a comparative study of one and two way traversers.

can convert our ρ^* to a distance-estimating form like h, where smallest is best. We had $\rho^* = \exp[\ -1.03\ -0.41f_1\ -0.83f_2\ -1.67f_3\ -0.47f_4\ ]$, so if we take the natural logarithm, then divide by 0.41 (the absolute value of the coefficient of the distance score term), we obtain a function Δ which can be used to estimate the distance from the goal (i.e. $h = \Delta$). For example, if, for a node A, $f_2(A) = f_3(A) = f_4(A) = 0$, then $\Delta(A) \leq$ the actual remaining distance to the goal. If $f_i(A) \neq 0$ for some $2 \leq i \leq 4$, we generally do not know whether $\Delta(A)$ underestimates the actual distance, but Δ should still be a good path length estimator. To complete the formation of our f, we simply take g(A) = level of A in the state tree.[1]

Using this $f = (1-w)g + w\Delta$, two experiments were conducted which were simplified versions of two of Gaschnig's extensive investigations, although each involved several test runs. In the first set, w was fixed at 1, and the upper bound d on the shortest solution length was varied. Unfortunately, this problem specification parameter is only an upper limit, unlike Gaschnig's for the eight puzzle (there is no practical way of determining the actual shortest solution length for harder instances of the fifteen puzzle). One set of 20 puzzle instances for each of 5 values of d was tested (each puzzle having been passed

[1] Recall from section 1.2 that strong conclusions can be drawn if h never overestimates the remaining path length, and if $w \leq 0.5$.

through a mild filter to eliminate some shorter than d). Although the abcissa -- real minimum path length -- is not accurate, the graph in figure 7.3 showing average number of developed nodes suggests an approximately linear relationship. This is similar to what Gaschnig found with a good evaluation function for the eight puzzle (no exponential explosion).

The second experiment again used $f = (1-w)g + w\Delta$ on a standard random set of 32 puzzle instances,[1] each presented for 7 different values of the control policy parameter w. As with our earlier performance measurements (subsection 7.2.1), we sometimes know just lower bounds on the average number of developed nodes, and only approximate values for solution lengths. The graph in figure 7.4 illustrates the variations. Note that the points are exact for $w=1$ and $w=0.95$, and otherwise accuracy decreases as w does -- the accuracy being a function of the number (also shown) of puzzle instances solved. Probably there is an increasing downward bias in solution length as w decreases, since there should be a tendency for shorter instances to be the ones remaining solvable (the data strengthen this suspicion).

There are a number of observations here. One is that the quality of the solutions improves (i.e. average solution lengths decrease) as w decreases. This is what is normally

[1] This was the same random set used to measure the performance of ρ^* originally.

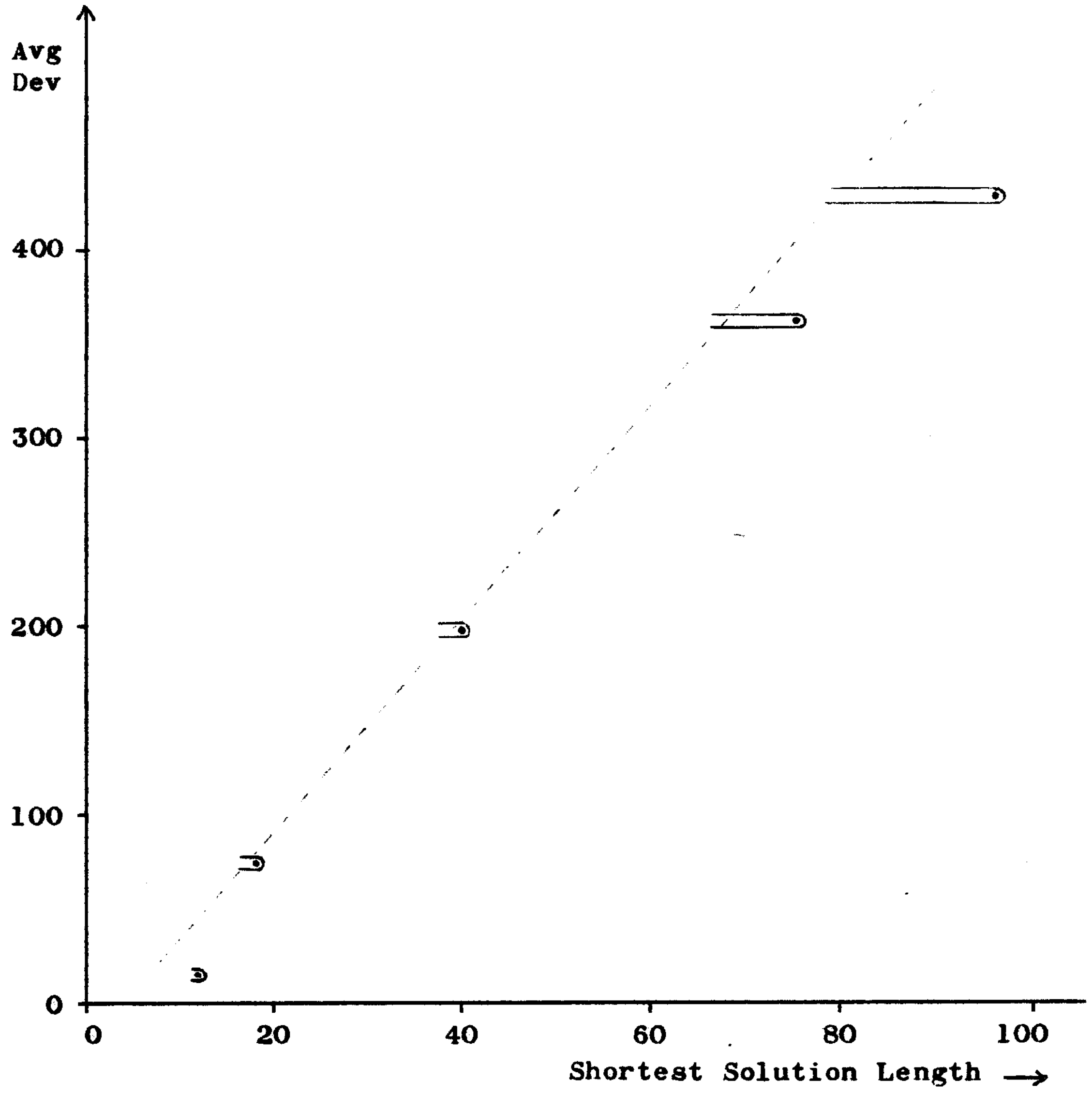


Figure 7.3. Graph of average number of nodes developed vs. minimum solution length for evaluation function Δ.

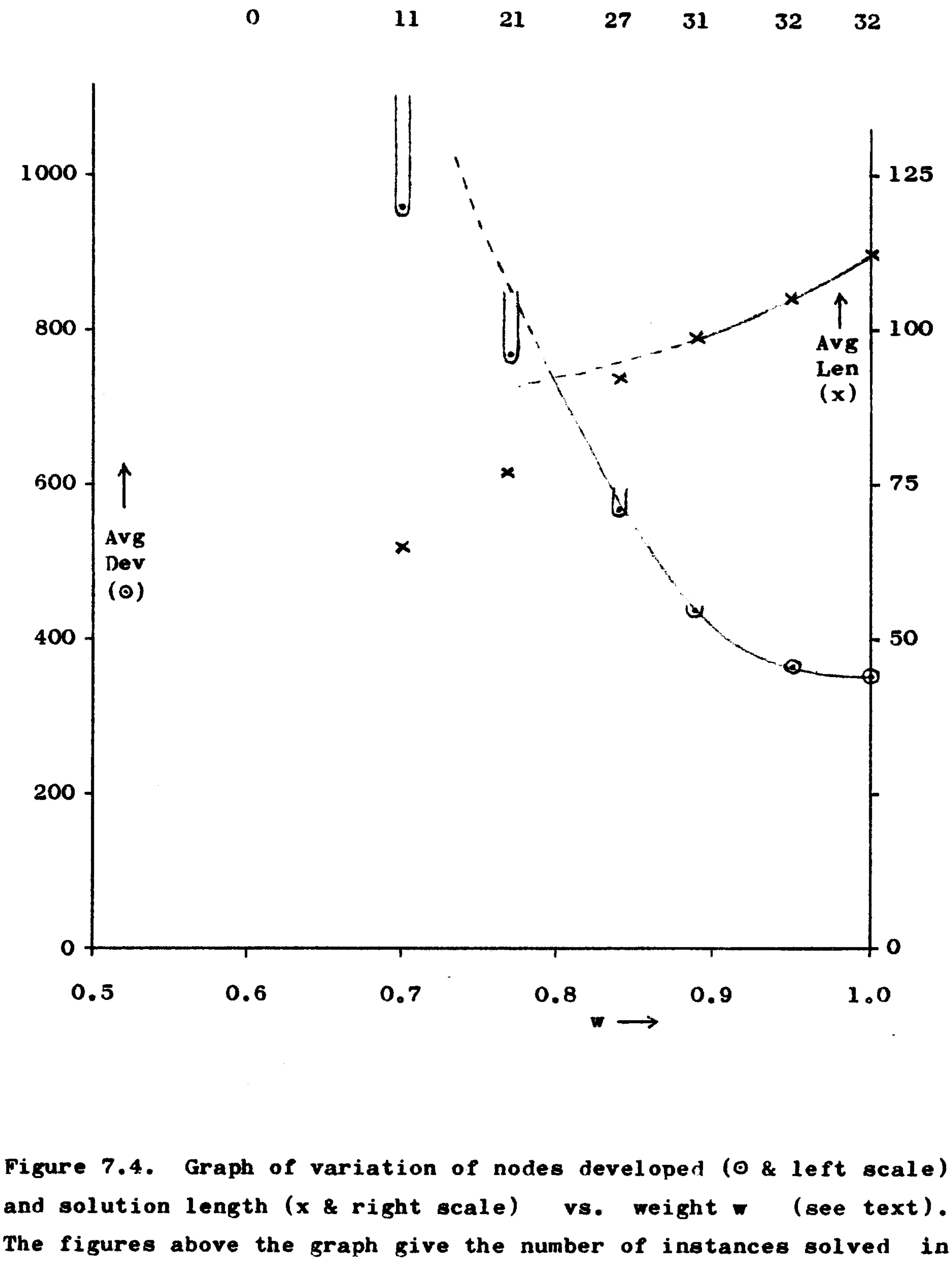


Figure 7.4. Graph of variation of nodes developed (⊙ & left scale) and solution length (x & right scale) vs. weight w (see text). The figures above the graph give the number of instances solved in a sample of 32.

found, since decreasing w increases the breadth-first element. (See Gaschnig's similar experimental results with the eight puzzle.)

Gaschnig's results showed that as the evaluation function improved (in terms of number of developed nodes before solution), it was aided less in terms of solution quality by decreasing w.[1] Our Δ is a fine evaluation function; it seems to benefit just slightly from a decrease in w.

Pohl's (1970) theoretical worst case analysis demonstrated that $w=0.5$ is at least as fast as $w=1$. However, in both Gaschnig's experiments and ours, $w=1$ is at least as fast as $w=0.5$ in <u>average</u> cases.

Another interesting aspect is that while Gaschnig discovered that for the best evaluation function he tested, increasing w beyond a certain value did not aid speed, our case contrasts: the higher w is, the fewer the number of nodes developed. This is in line with conjectures by Nilsson (1971) and Pohl (1970).

As Gaschnig pointed out, however, one must be cautious in making broad statements concerning general effects of varying control policy parameters such as w and h.

[1] Actually, the situation is more complex than this. For some of Gaschnig's tested functions, the speed variations with w depended on the difficulty of the problem instance.

8. CONCLUSIONS

This short chapter summarizes what has been learned about the system and mentions some possible avenues of further investigation.

8.1. SUMMARY OF PRESENT RESEARCH

In chapter three we hypothesized that localized true penetrance is a useful basis for evaluation functions. This has been verified; moreover, the methods for estimating true penetrance (chapter five) and of revising old estimates (chapter six) have also been shown to be effective, as has the method of finding a modelled penetrance function (chapter four).[1] At least for some feature sets, the system is well behaved; it distinguishes useful features, and the modelled evaluation function converges, to give good performance. And if the whole feature set is not congenial, it appears that interaction can allow the user to gain some helpful information about which individual features are useful and to what extent they are important (in two ways -- by observing splits, and by observing overall performance).

The plan seems more efficient or more reliable than alternative, less mechanized methods (chapter seven).

[1] An examination of the residuals of the regression step of experiment two (see the end of section 7.2.1) suggested that the log-linear model for the evaluation function is adequate, and also that the method of error weighting is reasonable.

Perhaps one of the mcst interesting aspects is that local optima can be found. Moreover, this occurs despite the fact that the plan does not use any feedback about the evaluation function in terms of its overall performance (such as number of problem instances solved or length of solution). Rather, the feedback is in the form of refined statistical measurements, localized in the attribute space. (This further validates true penetrance and its estimators.)

Furthermore, the localized performance measures may contribute to reliability and efficiency: The cumulative nature of the regions permits gradual improvement of the evaluation function. Their splitting into natural sizes and shapes allows meaningful use of penetrance information.

The plan seems characterizable as a variant of hill climbing, so local, rather than absolute optima would be expected.

In summary, it can be stated that the design appears promising, especially in view of several feasible refinements and extensions, some of which are briefly outlined in the following section.

8.2. FUTURE RESEARCH

The simplest continuation of this work is to investigate the behavior of the plan with other feature sets, and with different state-space problems, to determine which of the hypothesized properties generalize strongly. For example, under what conditions are locally optimal parameters found? And what happens when several important related features are present; can the best of them be distinguished, either automatically or interactively? Further, how do strongly interacting features generally influence the system?

Another straightforward experiment is to use the weighted evaluation function form $f = (1-w)g + wh$ instead of the purely heuristic one.

It would also be interesting to define features as functions of operators (and states), rather than solely of states, especially for problems such as formula manipulation.

Another approach in the system development might be to attempt improvement of some individual details of the design. For example, a different data structure for attribute vectors would allow a better representation for the center of gravity of a rectangle (for the regression). If the attribute space points were ordered in each dimension, it would be easy to design a fast algorithm which uses the total counts to give a weighted center point. (This would replace the inefficient SHRINK algorithm.) This data structure would also facilitate an improved clustering algorithm. Although the desirability of such sophistications would depend on the tradeoff between computational cost and the improvement resulting from their implementation, this particular alteration seems advisable.

Another possibility is to sophisticate the method of adjusting elementary penetrance values to estimate true ones. Frequently the smoothing (section 6.1) results in outliers; these may often simply be inaccurarate cumulative regions. Such data could be treated differently, e.g. they could be ignored in the smoothing and perhaps their error estimates increased temporarily. If it were later established that some such region were not spurious (i.e if the trend continued), it could then be trusted.

The facility already exists for extending the attribute space into higher dimensions at the start of any iteration; it might be useful in combination with a procedure which utilizes the strategy-power factors and power exponent to determine which areas of the attribute space need further differentiation. (Once this had been established, a new potentially useful feature could be added.) [1]

Considerable difficulty is anticipated in cases where there are marked non-linearities and interactions among important features. The problem of anomalous penetrance then becomes severe, and perhaps incapacitating, for the present system. (In many cases, higher order models would almost certainly be too unstable, considering the sparsity

[1] Other variations of current designs might include a two-way graph traverser, non-rectangular regions, recombination of regions, improved error estimates, etc.

of available data (regions) and their inaccuracy.) There are some subtleties that can be implemented in an attempt to meet this challenge, however. One simple modification is to insist that one true penetrance estimate is the same as another, unless their errors do not overlap. This might mean a longer convergence time, but probably greater accuracy. It, alone, is unlikely to be adequate though. Another scheme would approximate a non-linear model locally: For each region R, weight the other regions according to attribute space proximity to R (e.g. inverse-square Euclidean distance), then fit a linear model (using the stepwise regression procedure). There would then be a modelled penetrance function ρ_R for each region R, and ρ_R would be used when a state maps into R, or the same "proximity" relation could determine a weighted evaluation, using all the ρ_R simultaneously. If this plan extension were successful, it would constitute a standardized and completely automated method for accommodation of general feature interactions.

BIBLIOGRAPHY

Anderberg, M.R. (1973): Cluster Analysis for Applications, Academic Press.

Arbib, M.A. (1972): The Metaphorical Brain: An Introduction to Cybernetics as Artificial Intelligence and Brain Theory, Wiley.

Ball, W. (1931): Mathematical Recreations and Essays, Macmillan.

Bartlett, M.S. (1947): "The Use of Transformations," Biometrics, 3, 39-52.

Bindra, D. (1976): A Theory of Intelligent Behavior, Wiley.

Bruner, J.S., Goodnow, J.J. & Austin, G.A. (1956): A Study of Thinking, Wiley.

Buchanan, B.G., Johnson, C.R., Mitchell, T.M., & Smith, R.G. (1979): Models of Learning Systems, Stanford Artificial Intelligence Memo.

Chandra, A.K. (1972): (Stanford University Program).

Deese, J.E. & Hulse, S.H. (1967): The Psychology of Learning, McGraw-Hill.

Doran, J. (1967): "An Approach to Automatic Problem-Solving," Machine Intelligence 1 (eds. Collins, N.I. & Michie, D.), American Elsevier, pp. 105-123.

----- (1968): "New Developments of the Graph Traverser," Machine Intelligence 2 (eds. Dale, D. & Michie, D.), American Elsevier, pp. 119-135.

----- & Michie, D. (1966): "Experiments with the Graph Traverser Program," Proc. Roy. Soc., A, vol. 294, pp. 235-259.

Draper, N.R. & Smith, H. (1966): Applied Regression Analysis, Wiley.

Duda, R.O. & Hart, P.J. (1973): Pattern Classification and Scene Analysis, Wiley.

Finney, D.J. (1971): Probit Analysis, Cambridge University Press.

Fraser, D.A.S. (1958): Statistics -- An Introduction, Wiley.

Gardner, M. (1964): "Mathematical Games," _Scientific American_, vol. 212, no. 3, pp. 112-117.

Gibson, E.J. (1969): _Principles of Perceptual Learning and Development_, Appleton-Century-Crofts.

Hart, P., Nilsson, N.J. & Raphael, B. (1968): "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," _IEEE Trans. Sys. Sci. and Cybernetics_, vol. SSC-4, no. 2, pp. 100-107.

Hartigan, J.A. (1975): _Clustering Algorithms_, Wiley.

Holland, J.H. (1970): "Hierarchical Descriptions, Universal Spaces, and Adaptive Systems," _Essays on Cellular Automata_ (ed. Burks, A.W.), University of Illinois Press, pp. 320-353.

----- (1975): _Adaptation in Natural and Artificial Systems_, University of Michigan Press.

Huet, G.P. (1971): _Experiments with an Interactive Prover for Logic with Equality_, Report 1106, Case Western Reserve University.

Hunt, E.B. (1975): _Artificial Intelligence_, Academic Press.

Jackson, P.C. (1974): _Introduction to Artificial Intelligence_, Petrocelli.

Johnson, W.W. & Story, W.E. (1879): "Notes on the '15' Puzzle," _Amer. J. Math._, vol. 2, pp. 397-404.

Koestler, A. (1964): _The Act of Creation_, Dell. (Some other editions do not contain the important book two.)

Michie, D. (1967): "Strategy-Building with the Graph Traverser," _Machine Intelligence 1_ (eds. Collins, N.L. & Michie, D.), American Elsevier, pp. 135-152.

----- & Ross, R. (1970): "Experiments with the Adaptive Graph Traverser," _Machine Intelligence 5_ (eds. Meltzer, B. & Michie, D.), American Elsevier, pp. 301-318.

Minsky, M. & Papert, S. (1969): _Perceptrons_, MIT Press.

Newell, A. & Simon, H.A. (1972): _Human Problem Solving_, Prentice-Hall.

Nilsson, N.J. (1971): _Problem Solving Methods in Artificial Intelligence_, McGraw-Hill.

----- (1980): _Principles of Artificial Intelligence_, Tioga.

Pohl, I. (1969): Bidirectional and Heuristic Search in Path Problems, Report #104, Stanford Linear Accelerator Center, Stanford.

----- (1970): "First Results on the Effect of Error in Heuristic Search," Machine Intelligence 5 (eds. Meltzer, B. & Michie, D.), American Elsevier, pp. 219-236.

----- (1977): "Practical and Theoretical Considerations in Heuristic Search," Machine Intelligence 8 (eds. Elcock, E. & Michie, D.), Harwood, pp. 55-72.

Popplestone, R.J. (1967): Machine Intelligence 1 (eds. Collins, N.L. & Michie, D.), American Elsevier, pp. 31-46.

Rendell, L.A. (1977): A Locally Optimal Solution of the Fifteen Puzzle Produced by an Automatic Evaluation Function Generator, Research Report CS-77-36, Dept. of Computer Science, University of Waterloo.

Robinson, G. & Wos,L. (1969): "Paramodulation and Theorem-Proving in First-Order Theories with Equality," Machine Intelligence 4 (eds. Meltzer, B. & Michie, D.), American Elsevier.

Rosenblatt, F. (1962): Principles of Neurodynamics, Spartan.

Samuel, A.L. (1959): "Some Studies in Machine Learning Using the Game of Checkers," Computers and Thought (eds. Feigenbaum, E.A. & Feldman, J.), McGraw-Hill, pp. 71-105.

----- (1967): "Some Studies in Machine Learning Using the Game of Checkers II -- Recent Progress," IBM J. Res. and Develop., vol. 11, no. 6, pp. 601-617.

Schoenfield, J.R. (1967): Mathematical Logic, Addison-Wesley.

Schofield, P.D.A. (1967): "Complete Solution of the 'Eight Puzzle'," Machine Intelligence 1 (eds. Collins, N.L. & Michie, D.), American Elsevier, pp. 125-133.

Selfridge, O.G. (1958): "Pandemonium: A Paradigm for Learning," Pattern Recognition (ed. Uhr, L.), Wiley, pp. 339-348.

Simon, H.A. (1962): "The Architecture of Complexity," Am. Phil. Soc., vol. 106, no. 6, pp. 467-482.

Slagle, J.R. & Bursky, P. (1968): "Experiments with a Multipurpose, Theorem-Proving Heuristic Program," JACM, vol. 15, no. 1, pp. 85-99.

Slagle, J.R. & Farrel, C. (1971): "Experiments in Automatic Learning for a Multipurpose Heuristic Program," Comm. ACM, vol. 14, no. 12, pp. 91-99.

Snedecor, G.W. & Cochran, W.G. (1972): Statistical Methods, Iowa State University Press.

VanEmden, M.H. (1970): "Hierarchical Decomposition of Complexity," Machine Intelligence 5 (eds. Meltzer, B. & Michie, D.), pp. 361-380.

Werner, H. (1957): Comparitive Psychology of Mental Development, International Univerity Press.

Winston, P.H. (1977): Artificial Intelligence, Addison-Wesley.

INDEX OF DEFINITIONS