DETAILS OF AN AUTOMATIC EVALUATION FUNCTION GENERATOR FOR STATE-SPACE PROBLEMS

by

Larry Rendell
Department of Computer Science
University of Waterloo
Waterloo, Ontario

CS-78-38
August, 1978

Faculty of Mathematics

University of Waterloo
Waterloo, Ontario, Canada

ABSTRACT

Details are given of a system which automatically generates evaluation functions for state-space problems and which has created a function that solves the fifteen puzzle with (locally) optimal parameters. Problem instances are presented in order of (roughly) increasing difficulty; initially, at least one problem instance must be solvable breadth-first. Also at the outset, the system must be presented with an ordered set of features (functions mapping states into integers) as input. This ordered set of features defines a feature space.

The system is an iterative one, and each iteration consists of three steps: a solving step, a region (cluster) handling step, and a regression step. After a graph traverser attempts a set of problem instances (solving step), the system procedes to cluster probability estimates in the feature space, via an effective splitting algorithm (region handling step). From the clusters, parameters for a (not necessarily) linear evaluation function are computed (regression step). In post-initial iterations, the system's graph traverser utilizes an evaluation function generated by the preceding iteration, and, in these later stages, the region handling step refines established clusters both by revising previous probability estimates, and also by further splitting, to effect successively better evaluation functions.

PREFACE

This report presents the formal details of a system which has created an evaluation function that solves the fifteen puzzle. The evaluation function was found to have parameters which are locally optimal. These results are given in [Rendell].

Other experimental results will appear in the writer's PhD thesis. Also included in that paper will be a further examination of the properties of the system, and of the motivation for its design.

The basic ideas behind the system presented in this paper are intuitively simple, but considerable detail is necessary to formalize the methods. The introduction is therefore designed to give the reader a first impression of the approach. For those less familiar with the state space / feature space paradigm (or even mechanized problem solving), Appendix A should provide some illumination.

This paper contains an index of definitions.

1. INTRODUCTION

An overview of the iterative system is given below. Some simplifications are made, and there are omissions; the purpose at this point is just to familiarize the reader with the overall approach. (Appendix A provides a background for essential state-space terminology, and a very brief introduction to the system from a slightly different point of view.)

1.1. SOLVING STEP

The first step (presently) incorporates a one-way graph traverser and attempts to solve an input set of problem instances. Initially, this is done breadth-first, 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" of states results, which is just a record of the solution attempt. Nodes (states) are linked by immediate ancestor/offspring arcs. 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.

Ignoring arcs, each node of every final state graph is mapped into a point in the feature space. Although the

total number of possible points in the space might be very large, the density of points which comprise a current set may not be high; obviously there can be no more points than nodes in all the state graphs. Each point in the current set has a pair of integers associated with it. The "total count" (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", which is like the total count, except that it is further restricted to include only those nodes which appeared on a solution path (if any).

Now, the ratio good count / total count is the probability that a corresponding state was used in a solution of some problem instance. Unless both the extent and the dimensionality of the feature space are very small, the points will tend to have low counts, but let us ignore this fact for the moment. Basically, we shall assume that this probability, good count / total count, is a measure of the "goodness" of a feature space point for other problem instances, and we shall call it the "elementary usefulness" (of a point, for a particular problem instance set and evaluation function). The assumption of general applicability is presumptuous, since we shall in fact be generalizing from simpler to harder problem instances; but constant feedback and revision tend to correct biases.

In the section which follows, we shall elaborate on this elementary usefulness, as we outline how to cluster

points and their counts according to their feature space proximity, similarity of usefulness, and a reliability estimate.

1.2. REGION HANDLING STEP

This second step of an iteration uses the feature space points and their associated counts of the solving step either for clustering (first iteration), or for revising 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 feature space area, rather than just to a single point. Thus the elementary usefulness of a cluster (for a particular problem instance set and evaluation function) is its good count divided by its total count. And so, within its boundaries, a cluster represents a constant usefulness; it indicates the probability estimate of a corresponding state's being "good for a solution". (Because of this, the variation of usefulness with a feature should perhaps not be too erratic in practice).

In addition to the usefulness itself, it will be desirable to know how reliable the value is. One source of error is a random element which relates to the magnitudes of the counts. For example, a usefulness 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 an error term derived from this source, there are others which will be given later. For our present purposes it is enough to know that we can estimate a combined (usefulness) error (for a cluster). Generally, the combined cummulalative errors can be quite large, sometimes several times the usefulness itself. The errors are expressed as factors of the usefulness.

Clusters are restricted to be rectangular, with edges parallel to the axes, so little information is required for their specification (just the extreme corner points). The actual algorithm used for the clustering is a splitting algorithm. It inputs some set of points with their associated counts, as well as a rectangle which is aligned with the axes and surrounds all the points. Next, the algorithm tentatively splits the whole cluster into two rectangles, in every possible way (using every division in each feature 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; this distance is non-metric and defined in terms of the ratio of their usefulness, taking into account the usefulness error:

$$\text{distance}(r_1, r_2) = \frac{u_1/e_1}{u_2 \cdot e_2}$$

where $r_1 \cup r_2$ is the whole rectangle, u_i is the usefulness of r_i, e_i is the error for u_i $(i=1,2)$, and $u_1 \geq u_2$. Thus, the

best tentative split is the one such that the two rectangles are "most assuredly dissimilar" with regard to usefulness.

If this largest distance is less than unity, the two rectangles are recombined. If, however, the distance is greater than one, 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 set of clusters whose rectangles constitute a partition of the input rectangle, and whose counts define both an elementary usefulness and an error estimate for their rectangle.

Notice that the counts play a multiple role. They define the usefulness, partially determine the error, and thus govern the extent of splitting. The final number of clusters is largely determined by the counts (data), not by the algorithm alone.

Generally, to the extent that a feature is "useful", there will be splitting in that dimension.

Chapter three discusses count functions, usefulness probabilities and errors, and chapter five details the clustering process.

Now that we have examined the splitting algorithm, let us see how it is used.

From iteration to iteration, we shall find it appropriate to keep track of the usefulness of a cluster,

rather than to retain the counts. So we define a "region" to be a rectangle (aligned with the axes), together with a pair of real numbers, the usefulness and the (usefulness) error. A set of regions (from past iterations), together with the set of feature space points and associated counts (from the solution step of the current iteration) constitute the input for the region handling step. For the very first iteration, there is only one input region -- that whose rectangle minimally encloses the points. For all later iterations, the input region set is that from the preceeding iteration.

For an initial iteration, the clustering algorithm is called just once, with the point enclosing rectangle as input. The solution searches from which the point/count set is generated are breadth-first for a first iteration, so the elementary usefulness values from the output clusters of the splitting algorithm become absolute usefulness values for a corresponding region set (and similarily for the error values).

For post-initial iterations, the clustering algorithm is called once for each established region. Suppose that a region input is R=(r,u,e) where r is its rectangle, u its usefulness, and e its error. And suppose that the algorithm splits r into m rectangles $r_1, r_2, \ldots, r_m$ whose counts are $(g_1,t_1),(g_2,t_2), \ldots, (g_m,t_m)$. Then each new region R_i ($i \leq m$) spawned from R will have a usefulness $(g_i/t_i).k$ where

$k = u/(\Sigma g_i / \Sigma t_i)$. (Generally, $k<<1$.)

The region handling step, for iterations after the first, not only refines established regions by further splitting, it also updates the usefulness estimate for each region. To revise established regions is not perfectly simple, as the following discussion attests.

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 feature space points lying within r_1 and r_2 are (g_1, t_1) and (g_2, t_2). Generally $g_i/t_i > u_i$, or else the heuristic is not working properly; for the u_i supposedly represent absolute probabilities whereas the counts reflect probabilities which are conditional on the particular search used. If g_i/t_i were equal to or less than u_i, then the number of states developed in order to find one used in a solution would be equal to or greater than the number if a breadth-first search had been used; thus harder problems could not be solved.

So, to revise u_i, we cannot (say) take the average of u_i and g_i/t_i, 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 upwards and/or u_2 downwards. The exact manner in which the system accomplishes this revision will be seen later. The new value of usefulness becomes a sum (weighted according to errors) of the old value and a corrected

current elementary usefulness value. Usefulness becomes a product of experience over all the iterations. This revision has an effect of decreasing errors and, of course, allows some readjustment of earlier usefulness estimates.

Chapter four defines regions; chapter five shows how the clustering algorithm is used in the first iteration; and chapter six describes the complex process of region revision in later iterations.

1.3. REGRESSION STEP AND EVALUATION FUNCTION

After the number of regions increases sufficiently, a third, regression step becomes part of each iteration. For this, the center point of each region provides the values for the n independent variables (feature values), while the associated usefulness is the dependent variable. These variable sets are weighted according to the accompanying usefulness error. (Recall that a rectangle, plus the usefulness and its estimated error constitute a region). A stepwise regression algorithm is used, and the models have (to date) been restricted to be linear, so the resulting number of non-zero terms or parameters can be from one (constant) to n+1.

The evaluation function for iteration I+1 uses both the regions from iteration I, and the regression model from iteration I. If a state maps to a feature space point $\underline{x}$,

then the valuation is a weighted combination of u and m($\underline{x}$) where R = (r,u,e) is the region enclosing $\underline{x}$, and m is the regression model. The weighting depends on the estimated errors. For the final evaluation function, m($\underline{x}$), alone, has been used.

Chapter four introduces the regression step and evaluation 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 results to improve the heuristic.

1.4. DESIGN PHILOSOPHY AND PREAMBLE

The main goal in this research was to develop a system which could operate at a relatively high abstract level (forming generalizations in a feature space) without severe quantitative restrictions, one that could be sophisticated but possibly "fuzzy" in detail. This is reflected, for example, in the definitions of error components; just order of magnitude values are required. And contributing error terms are summed directly (rather than adding variances as would be more strictly correct). There are many areas where alternative details could be substituted; the reader may perhaps think of possible improvements.

The reader may find the appendices useful. Appendix A, to which we have already referred, contains a summary of background material and a short introduction to this system for those less familiar with the state-space/feature space approach to problem solving. Appendix B describes estimation of a non-randomness error; and Appendix C discusses a standard notation for usefulness and its error. Definitions can be located in the text by use of the index.

Chapter two begins a detailed formal description.

2. PRELIMINARY DISCUSSION AND DEFINITIONS

The definitions of state-space problem, operator, graph traverser, search strategy, evaluation function, etc. are standard and can be found in [Nilsson] and elsewhere. Appendix A summarizes and illustrates some of these background concepts.

We shall now begin to lay the ground work for the formalization of the system by examining three distinct spaces which will be involved and by viewing some of the relationships among some elements in those spaces.

2.1. SPACES

In addition to a space of states, we shall be concerned with two other spaces. One is the ranking space which is just the one dimensional interval of real numbers: [0,1]. Our evaluation functions will map members of a state space into numbers in the ranking space, for the purpose of ordering states according to "usefulness" in a possible solution and thus for selection for development. We shall think of the real numbers in [0,1] as representing probabilities.

The other space is a feature space. Features, or attributes of states have frequently been used to abstract some measurable quality thought to relate to the "usefulness" of a state. Often an evaluation function has been

specified as a linear combination of features [Slagle & Farrel, Doran & Michie].[1] More generally, we can think of a feature space as being defined by an ordered set of features. We then have the problem of meanfully mapping points or areas in the feature space into points in the ranking space. Our evaluation function need not necessarily be linear nor continuous [Michie & Ross]. In fact the evaluation functions we shall meet here relate to regions of feature spaces; we shall be clustering points in the feature space to form local groups of reasonably uniform "usefulness".

Throughout this paper, we shall deal with feature spaces, all of which will be integer; i.e. features will be restricted to have integer values only.

To summarize this overview in terms of spaces, then, states can be mapped to feature spaces, and feature space points or areas to probability estimates in a ranking space, so that a given state can be evaluated according to its feature space correspondence and resulting ranking space probability value --- the "usefulness".

2.2. SOME DEFINITIONS AND EXPLICIT RELATIONSHIPS

Suppose we have a problem instance, P, which consists of a starting state, goal state, and set of operators.

[1] Features have also been commonly used in pattern recognition.

Suppose, also that we have a graph traverser which is designed to cease working on a problem after a given number of states has been generated. An explicit graph, corresponding to P, with states as nodes, operators as arcs, which has been built up at any point in the operation of the graph traverser, we can call simply a state graph. Eventually, either the graph traverser finds a solution or else it gives up. We shall call the state graph which results in either of these two situations, a final state graph.

Suppose that G is a final state graph, and that π is a node in G. Let an ordered set of features be represented by $F = \{f_1, f_2, \ldots, f_n\}$. This set defines a feature space, F. A point in F corresponding to π is given by $(f_1(G,\pi), f_2(G,\pi), \ldots, f_n(G,\pi)) = \underline{f}(G,\pi)$ in vector notation. (Generally, we shall use $\underline{f}$ to denote the vector function $(f_1, f_2, \ldots, f_n)$.) The feature space point is a function of G if the feature depends on the history of π. Throughout this paper, only features which are independent of the structure of a state graph will be used, and we shall simply write $\underline{f}(\pi)$ to refer to the feature space map of π, or the point in F corresponding to π. We can also say that some feature space point or area in F has corresponding nodes (or states) in G. There may be zero, one, or a larger number of states in G corresponding to a point or area of F.

Suppose that θ is an evaluation function which maps a

set of states into [0,1] (state-space to ranking space). Let P be a problem instance. Then the final state graph, G (for P and θ) is the final state graph created by a graph traverser using θ as its guide in attempting to solve P. We can write $G = G(\theta, P)$ and abbreviate to G if the arguments are understood. We shall often refer to sets of problem instances, $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ and corresponding sets of final state graphs, $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$, with $G_i = G_i(\theta, P_i)$, or $\mathcal{G} = \mathcal{G}(\theta, \mathcal{P})$. If $\mathcal{G} = \mathcal{G}(\theta, \mathcal{P})$, we can say that $\mathcal{G}$ is the final state graph set (for $\mathcal{P}$ and θ), that $\mathcal{P}$ and θ determine $\mathcal{G}$, and that $\mathcal{G}$, θ, and $\mathcal{P}$ are associated.

2.3. A FIRST VIEW OF THE ITERATIVE SYSTEM

Each iteration takes place in three steps. The first is the solving step, the second is the region handling step, and the third is the regression step. The second and third steps will be detailed later on, but the first can be described now. The solving step always begins with a given evaluation function and problem instance set, and determines the associated final state graph set.

We frequently shall use the subscript I to refer to iteration I. The problem instance set $\mathcal{P}_I$, which determines the final state graph set, $\mathcal{G}_I$, is called the training (problem) set (for iteration I). The evaluation function

which is always associated with P_I and G_I is θ_{I-1}, i.e. the one from the preceding iteration. Thus $G_I = G(\theta_{I-1}, P_I)$. G_I is the final state graph set of (for) iteration I. G represents the determination of G_I from its two arguments. θ_0 is a constant.[1]

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

The determination of G_I from θ_{I-1} and P_I is called the first step of iteration I or solving step of iteration I. Later on, we shall encounter the second step, which forms or revises regions in a feature space, and the third step, which computes an evaluation function, θ_I, from these regions.

2.4. THE STANDARD ILLUSTRATIVE EXAMPLE

In order to clarify definitions, examples appear throughout this paper. Every example chosen was from an actual single iterative series, an experiment with the fifteen puzzle. The feature set for this particular experiment contained four features, which were:

[1] In experiments to date. But not necessarily.

f_1 - distance score (sum of distances of each tile from "home")

f_2 - order wrong score (Examine each line, i.e. row or column and count one for each occurrence of two tiles being in their proper line, but out of order.)

f_3 - line wrong score (Examine each line and count one if all the tiles of the line are the ones that should be in that line, but they are not in the correct order.)

f_4 - blocked score (Examine each line and count one for each occurrence of two tiles being in their correct place, but with an alien tile intervening.)

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 feature space was about 4000. Generally, the training problem sets, P_I, had a dozen or fewer members; the associated final state graphs, G_I, had a few thousand nodes, but the feature state maps of G_I never included more than a few hundred points.

3. CLUSTERS AND PROBABILITY ESTIMATES

As stated in the previous chapter, we would like to map states into points in a feature space, and there form clusters. The cluster boundaries should be chosen so that all points within a cluster correspond to states which have roughly the same "probability of being useful in a solution" (being on a "good" solution path). Thus a single "probability" can be assigned to a cluster. In later chapters we shall see just how these clusters, each with its accompanying "usefulness probabilty", can actually be formed, but in this chapter we shall concentrate on the clusters themselves.

3.1. RECTANGULAR CLUSTERS

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

We define: A feature space is an integer feature

space. A rectangle is the usual entity, but always in some feature space and aligned with the feature space axes. lp and up refer to the n dimensional lower and upper corner point vectors, respectively. If r is a rectangle, lp(r) and up(r) are its extreme corner points.

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

Later on, we shall see how the actual clusters can be created; their boundaries and number being determined by the data. At this time we introduce an important pair of "count" functions. These functions have a dual purpose; they not only will be used to construct and modify the emerging evaluation function, but also they will participate centrally in the process of clustering. Both of these roles will gradually become clearer.

3.2. COUNT FUNCTIONS

Suppose we have the following:

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

(2) A feature space, F, defined by the ordered set of n features, $F = \{f_1, f_2, \text{---}, f_n\}$.

(3) A rectangle, r, in F.

We define the two count functions, g, and t: The total (state) count of r (for F and G),

$t(F, G, r)$ $\overline{\text{def}}$ the total number of developed nodes, π, in every final state graph, $G \in G$, such that $\underline{f}(\pi) \in r$.

And the good (state) count of r (for F and G),

$g(F, G, r)$ $\overline{\text{def}}$ the number of developed nodes, π, in all $G \in G$, such that:

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

(2) π participates in the solution (if any).

If a particular feature space is understood, we can abbreviate the counts to $g(G, r)$ and $t(G, r)$.

Now let us define a probability estimate based on the ratio of the two counts.

3.3. PROBABILTIY ESTIMATES OR USEFULNESS VALUES

The elementary usefulness of a rectangle, r, (for a set, G, of final state graphs, and feature set, F),

$$u(F, G, r) \underset{\text{def}}{=} \begin{cases} g(F, G, r)/t(F, G, r), & \text{if } g(F, G, r) \neq 0 \\ zval/t(F, G, r), & \text{if } g(F, G, r) = 0. \end{cases}$$ [1]

If we are given a state, π, randomly selected from any graph G of the set G ($\pi \in G \in G$), whose feature space map falls within a rectangle, r ($\underline{f}(\pi) \in r$), then the value, $u(G, r)$, is precisely the probability that π lay on one of the solution paths of G. This elementary usefulness is simply the ratio of the number of "good" states to the total number of states (in G), so that if $u(G, r)$ is close to unity then r is a "good" rectangle, indicating a high likelihood of a corresponding state's being "useful" in a solution (in G).

What we shall do is to generalize these statistics from G to estimate the "goodness" of states which arise in other problem solution attempts. The elementary usefulness $u(G, r)$ will be used to evaluate states corresponding to r, not only for different problem instances, but even for varied search strategies. (Recall that $G = G(\theta, P)$.) The

[1] Zval is a system parameter which has a value between zero and one. It is used to differentiate between, for example, 0/10 and 0/100. A typical value of zval is 0.5.

section following this contains further comment about the ramifications of our generalization. And in the next chapter we shall see exactly how our generalization is effected. Stil later, we shall use a more sophisticated usefulness, in which more than just the elementary usefulness will play a part, although the count functions will remain central.

Example 3.3

Consider our standard example which has a feature space of four dimensions (see the preceding chapter). The particular results which we shall view now resulted from the breadth-first search of four easy puzzles, each nine moves away from the goal. Three clusters were formed[1], whose corner points, count function and usefulness values are given in table 3.3.

Table 3.3 Cluster Statistics

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

[1] The mechanism for cluster creation is given in chapter five.

[2] Zval = 0.5.

3.4. ERROR ESTIMATION

We now have estimates of the (elementary) usefulness of a cluster but we would also like a measurement of the reliability of this value, because we shall want (in chapter five) to know whether the usefulness of one rectangle is significantly different from that of a rectangle adjacent to it in the feature space. If the two usefulness figures differ greatly, the clusters should be disjoint; if not, they should merge. (There are also other reasons why we shall need to have an error estimate; these will be discovered in a later chapter.)

Suppose we have some final state graph, G, and rectangle r, with its count functions $g(G,r)$ and $t(G,r)$. (We assume some feature space, whose vector function is $\underline{f}$.) Let us abbreviate $g(G,r)$ and $t(G,r)$ by γ and τ, respectively. If we generalize to other problem instances and search strategies, there are two sources of error in the usefulness, $u(G,r) = \gamma/\tau$. One relates to the magnitudes of γ and τ. For example, 1/20 is less reliable than 10/200.

The other error source is one about which we know little: We have assumed that usefulness values for one final graph set, $G = G(\theta, P)$, reflect precisely those for other problem instances and search strategies. But in fact we shall apply usefulness estimates to increasingly more difficult problem instances, and it seems likely that, in

general, these values might differ for harder samples. We are not generalizing from one random set of problem instances to another, but rather from easier problems to harder ones. Furthermore, the fact that the search strategy (evaluation function) is dynamic additionally complicates the situation. This severe error source, we should expect to vary, too, both in size and quality, depending on the particular state-space problem.

The 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. For now, let us formalize 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 shall need only an approximation.

We have $\gamma = g(G,r)$ and $\tau = t(G,r)$. Consider the entire set, S, of all possible states reachable from all possible starting states of a given problem schema --- the whole state-space. Let us assume that the nodes which contributed to the count, τ, were chosen randomly from S, rather than by the selection mechanism, θ, for the particular problem solution attempt. Then, for the total count, τ, we have essentially a binomial distribution, whose standard deviation estimate is

$\sqrt{N(r)\ \frac{\tau}{N(r)}\left(1 - \frac{\tau}{N(r)}\right)}$, where N(r) is the number of states, π, of S so that $\underline{f}(\pi) \in r$. If N(r) is large compared with τ, the expression becomes approximately $\sqrt{\tau}$. A similar argument gives an estimate of the standard deviation of the good count, γ, as $\sqrt{\gamma}$.

Thus, if the good count and total count are altered by one standard deviation in the expression for the elementary usefulness, to give a higher value, and if this is divided by the original elementary usefulness, γ/τ, we obtain an "error factor" of

$$\frac{\gamma + \sqrt{\gamma}}{\tau - \sqrt{\tau}} \Big/ \frac{\gamma}{\tau} = \frac{1 + \frac{1}{\sqrt{\gamma}}}{1 - \frac{1}{\sqrt{\tau}}} \quad , \quad \text{if } \gamma \neq 0,\ \tau > 1.$$

Generally we shall record errors in this way, i.e. as "high side" estimates expressed as fractions of the corresponding random variable. We shall call these error factors simply "deviations" or "errors" (depending on whether a "confidence factor" is included or not). So we define: The count deviation (of the pair (γ, τ)),

$$\underline{\text{devc}}(\gamma, \tau) \overline{\text{def}} \quad \frac{1 + 1/\sqrt{\gamma'}}{1 - 1/\sqrt{\tau}}$$

where $\gamma' = \gamma$ if $\gamma \neq 0$ and $\gamma' = \text{zval}$ if $\gamma = 0$.

This is actually the deviation on the "high" side. A similar expression would give the "lower" deviation. Nevertheless, we shall call the upper deviation just the

deviation.

If we had alternately assumed a binomial distribution for the probability estimate, $u(G, r)$, itself, then we would have obtained an expression for the standard deviation, expressed as a fraction of the elementary usefulness, of:

$$\sqrt{\left(\frac{\gamma}{\tau}\right)\left(1 - \frac{\gamma}{\tau}\right)/\tau} \Big/ \frac{\gamma}{\tau} = \sqrt{\frac{1}{\gamma} - \frac{1}{\tau}} =$$

$$= \frac{\sqrt{\tau - \gamma}}{\sqrt{\gamma\tau}} \leq \frac{\sqrt{\tau + \gamma}}{\sqrt{\gamma\tau}} = \frac{1}{\sqrt{\gamma}} + \frac{1}{\sqrt{\tau}}$$

$$\leq \frac{1 + \frac{1}{\sqrt{\gamma}}}{1 - \frac{1}{\sqrt{\tau}}} - 1 = \text{devc}(\gamma,\tau) - 1.$$

Our deviation is a slight overestimate compared with $1 + \sqrt{1/\gamma - 1/\tau}$ (especially for cases in which γ is close to τ).

Let us consider the distribution of the total error, of which the count deviation is only a part. Often, when the variance is relatively small, the error is known or assumed to be normally distributed. Although it is not apparent at this stage, our total error estimates will regularly be very large; the error term we have just analized is only a beginning. In addition to this count deviation, there is

not only the non-randomness error which we have briefly mentioned, but there will also sometimes be accumulating errors as a result of the ongoing iterative process. Errors of a factor of ten will be typical. Thus the error distribution will generally be positivly skewed; an expression such as v + 10v does not make sense for the lower of the two extremes. This is the reason why the count deviation is, and all other deviations and errors will be, expressed as factors rather than as additive terms.

We shall therefore assume a log-normal distribution for both the usefulness and its error. This distribution is positively skewed, and the lower limit (zero) is correct, both for the usefulness and for its error, if we express log(usefulness) = log(v) $\pm$ c.log(e) where v is the usefulness estimate, e the deviation and c the <u>confidence factor</u> (number of standard deviations). So the upper "likely limit" of the usefulness is $v.e^c$ (or unity, whichever is lesser) and the lower limit is v/e^c.

If we were to ignore other error sources, we would simply substitute the count deviation for e, but generally we shall sum logarithms of various contributing errors to obtain a rough estimate of the total.

We shall use a standard notation for deviations. All (with one exception) will have the prefix "dev" with a suffix indicating the source of the error. We shall further prefix a deviation, with "ln", to denote the logarithm of

that deviation (the log-deviation). This allows us to write the logarithmic forms of expressions more simply. Another fixed notational device to be found throughout is the use of the letter sequence "err" in place of "dev", particularly for the logarithmic forms, in which the log-error equals the log-deviation multiplied by the confidence factor.

For example, lnerrc = c.lndevc. The user-defined error, erru is meant to be added to errc as an estimate of the non-random error due to generalization of statistics from easier problem instances to harder ones, and from one search strategy to another. (This error term is again encountered in a later chapter, and is discussed in Appendix B.) According to our convention, lnerru = log(erru) and lnerru = c.lndevu.

4. REGIONS, THE HEURISTIC, AND THE ITERATIVE SYSTEM

In this chapter, we define the evaluation function, and in order to do so, we first introduce a convenience, the "region", which groups together information about a feature space rectangle and its usefulness. The concept of the region also facilitates an outline of the entire iterative process, given toward the end of this 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 feature space, F. Define an unreduced region, R (in F), to be a five-tuple, $(r(R), \gamma(R), \tau(R), \kappa(R), \epsilon(R))$, where r(R) is the rectangle of R (in F), $\gamma(R)$ is the good count of R, $\tau(R)$ the total count of R, $\kappa(R)$ the multiplier of R, and $\epsilon(R)$ the multiplier error of R.

In use, the second and third elements of a region, R, $\gamma(R)$, and $\tau(R)$, are always $g(G, r(R))$ and $t(G, r(R))$ (the two count functions of the preceding chapter) for some final state graph set, G. The meanings of the last two elements, $\kappa(R)$ and $\epsilon(R)$ will not really be clear until chapter six,

but we can say now that they relate to the past history of a region. $\kappa(R)$ is a multiplier for $\gamma(R)/\tau(R)$, and will be used in a way so as to cause the product $\gamma(R)/\tau(R).\kappa(R)$ to indicate "true" usefulness, later on in an iterative series, when the elementary usefulness values no longer directly reflect absolute probabilities. We define a function over the set of unreduced regions in a feature space:

If R is an unreduced region, then the usefulness of R,

$$\underline{val}(R) \quad \overline{\underline{def}} \quad \begin{cases} \gamma(R)/\tau(R)).\kappa(R) , & \text{if } \gamma(R) \neq 0 \\ zval/\tau(R).\kappa(R) , & \text{if } \gamma(R) = 0. \end{cases}$$

And another function, the total error of R,

$$\underline{err}(R) \quad \overline{\underline{def}} \quad \exp\,[c.(lndevc(\gamma(R), \tau(R)) + lndevu(R)) + ln\epsilon(R)] ,$$

where devc is the count deviation of chapter three, $ln\epsilon(R) = \log(\epsilon(R))$, and devu is the deviation component supplied by the user of the system. (See Appendix B.) The constant c is the confidence factor.

In keeping with the standard notation for deviations and errors, lnerr(R) is just the expression inside the square brackets above, lndev(R) is lnerr(R)/c, etc.

4.1.2. REDUCED REGIONS

If R is an unreduced region, then the reduced region (of R) is the triple,

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

The usefulness of a reduced region is its second element and the total error of a reduced region is the third element.

We shall refer to red(R) simply as R when the distinction is unimportant. For example the usefulness and total error mean the same thing for reduced regions as they do for unreduced regions, and the ambiguous use of val and err as functions over unreduced regions and as reduced region element references will present no problem.

Example 4.1

Example 3.3 showed three different rectangles and their count functions. In region formulation with unity multipliers and unity multiplier errors, 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 usefulness and total error (if devu = 1 and c=1) are:

$val(R_1) = 0.25$ $\qquad$ $err(R_1) = 1.38$

$val(R_2) = 0.016$ $\qquad$ $err(R_2) = 2.94$

$val(R_3) = 0.0002$ $\qquad$ $err(R_3) = 2.46$

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

If devu(R) is defined as $1 + 1/\sqrt{50val(R)}$ (Appendix B) then the regions are: (r_1, 0.25, 1.77); (r_2, 0.016, 6.2); (r_3, 0.0002, 25.).

4.2. THE EVALUATION FUNCTION

As we shall see later in this chapter, there is a certain set of regions, which will be termed "cummulative" regions, that is modified at the second step of each iteration. These special regions contain the entirety of heuristic information about the state-space problem and feature space to which they refer, and it is by these regions that we form our evaluation function. We shall now proceed to define the evaluation function, and later on discuss exactly how the cummulative regions are created and revised.

Since cummulative regions are altered once for each complete iteration, the evaluation function is dynamic too, and is recreated at the third step of an iteration. This function, the "valuation function", is composed of two parts; one is defined simply and directly from the regions ("region valuation function"), and the other is also a product of the cummulative regions, but indirectly through a regression ("regression valuation function").

First the direct component:

4.2.1. REGION VALUATION FUNCTION

Suppose we have some state-space problem, a state π of a state graph, a feature set $F = \{ f_1, f_2, \ldots, f_n \}$, and a set of regions, $R = \{ R_1, R_2, \ldots, R_m \}$ in the feature space, F, defined by F. We define the region-usefulness of π (for R),

$$\nu(R, \pi) \overline{\text{def}} \begin{cases} \text{val}(R), & \text{if } \underline{f}(\pi) \; r(R),\ R \in R \\ \max_{Q \in \mathcal{Q}}(\text{val}(Q)), & \text{if } \underline{f}(\pi) \notin r(R),\ \forall R \in R, \end{cases}$$

where $\mathcal{Q} = \{ Q \in R \mid \forall R \in R,\ ||\underline{f}(\pi) - r(Q)|| \leq ||\underline{f}(\pi) - r(R)|| \}$.[1]

If R is understood, we can abbreviate to $\nu(\pi)$. ν is the region valuation function.

We also define the error of the the region valuation (for π and R) in the obvious way:

$\underline{\text{err}}_\nu(R, \pi) \overline{\text{def}}$ err(R), where R is the region into which $\underline{f}(\pi)$ falls or, failing enclosure, the region which satisfies the above expression, for which val(R) is a maximum of the "closest" regions, $\mathcal{Q}$. $\text{Err}_\nu(R, \pi)$ can be referenced just as $\text{err}^\nu(\pi)$, if R is understood.

[1] $||\ ||$ represents Euclidean distance (from the feature space point to the closest point of the rectangle).

4.2.2. REGRESSION VALUATION FUNCTION

The other component of the evaluation function is obtained by regressing the natural logarithm of the usefulness of regions on their center points. The logarithm of the usefulness, rather than the usefulness itself, was selected because the relationship between the usefulness and a feature often seems to be close to logarithmic, and a uniform treatment was desired for features in general. Also, with a logarithmic formulation, there can be no problem with negative predicted values.

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 a forward stepwise (with a backward glance) selection procedure was chosen. This algorithm is described in [Draper & Smith] and implemented under the International Mathematical and Statistical (Fortran) Library (IMSL) name of RLSTEP.

One of the parameters of RLSTEP is the confidence level, α. So this now becomes one of the system parameters.

We can now formalize the details of the regression. We have previously defined the vector functions lp and up to refer to the extreme corner points of a rectangle. 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,

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

Suppose again that we have a state-space problem, a feature set, $F = \{f_1, f_2, \ldots, f_n\}$ and space F, and a set of regions, $R = \{R_1, R_2, \ldots, R_m\}$. Let $\underline{c}_j = \underline{cp}(r(R_j))$ and $y_j = \log(val(R_j))$. Let $\underline{z} = (x_1, x_2, \ldots, x_n)$ be a vector in F. Form the augmented vector of $\underline{z}$, $\underline{x} = h(\underline{z}) = (1, x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_q)$ where x_i is the feature space coordinate in dimension i, for $i \leq n$, and $x_i = x_k^p$ for $i > n$, where $k < n$ and $p > 1$. This will allow higher degree regression for which h is the augmenting function.[1]

Using the $\underline{c}_j$ as m observed value sets for the q independent variables, in the augmented vector $h(\underline{c}_j)$, and with the y_j as the corresponding dependent variables, we obtain: $JX'V^{-1}X\underline{a} = JX'V^{-1}\underline{y}$ where X is the mx(q+1) matrix composed of the m augmented vectors plus a column of 1's; $\underline{y}$ is the vector comprised by the m y_j values; $\underline{a}$ is the parameter vector (with q+1 elements); V is the mxm variance matrix which weights the observations; and J is the qxq selection matrix returned by RLSTEP (with 1's and 0's on the main diagonal and 0's elsewhere).[2]

The only element in this equation which has not been

[1] For all experiments to date, q=n, i.e. the models were all linear.

[2] See [Draper & Smith], chapter two.

specified is the variance matrix, V. If we assume that the observations are independent, then only the main diagonal of V is non-zero. We estimate these variances by the squares of the $\text{lnerr}(R_j)$, which are proportional to estimates of the standard deviations (see subsection 4.1.1). In other words, each region counts an amount which is weighted by the inverse of the logarithm of its estimated error.[1]

If $\hat{y}$ is a predicted value of the logarithm of the usefulness, then we have $\hat{y} = \underline{a}.\underline{x} = \underline{a}.h(\underline{z})$. So if π is a state, the predicted usefulness of π (for the region set R and confidence level α),

$$\rho(R,\alpha,\pi) \overline{\text{def}} \exp[\underline{a} \cdot h(\underline{f}(\pi))]$$

$$= \exp[\underline{a} \cdot (1,\underline{f}(\pi))], \text{ if the model is linear.}$$

If α and R are understood, we can write just $\rho(\pi)$. ρ is the regression valuation function.

Note that $\underline{a} = \underline{a}(R,\alpha)$.

We shall later have reason to refer to a related function, over a set of regions. If R is a region, then the predicted usefulness of R (for R and α),

$$\rho\text{val}(R,\alpha,R) \overline{\text{def}} \exp[\underline{a}(R,\alpha) \cdot h(\underline{cp}(r(R)))] .$$

The variance of y is given by $X(JX'V^{-1}X)^{-1}\sigma^2X'$, and σ^2 is estimated by the regression residual sum of squares, s^2. So we define the log-regression error (for π, R and α),

[1] See also subsection 6.1.4.

$$\ln\mathrm{err}_\rho(R, \alpha, \pi) \overline{\mathrm{def}} \; cX(JX'V^{-1}X)^{-1}s^2X.$$

(Recall that c is the confidence factor.) According to our convention for errors, deviations, and their logarithms, $\mathrm{err}_\rho(\pi) = \exp[\ln\mathrm{err}_\rho(\pi)]$, etc.

4.2.3. THE AVERAGE VALUATION FUNCTION

Finally, we combine the region valuation function with the regression one to form the (average) usefulness of a state π (for R and α),

$$\theta(R,\alpha,\pi) \overline{\mathrm{def}} \begin{cases} \exp\left[\dfrac{\dfrac{\log(\nu(\pi))}{(\ln\mathrm{err}_\nu(\pi))^2} + \dfrac{\log(\rho(\pi))}{(\ln\mathrm{err}_\rho(\pi))^2}}{\dfrac{1}{(\ln\mathrm{err}_\nu(\pi))^2} + \dfrac{1}{(\ln\mathrm{err}_\rho(\pi))^2}}\right], & \text{if } R \text{ has more than } ||J|| \text{ members}^{1}, \\ \nu(\pi), & \text{if } R \text{ has between one and } ||J|| \text{ members}^{1}, \\ 0.5, & \text{if } R = \phi. \end{cases}$$

[1] $||J||$ gives the number of non-zero elements on the main diagonal.

θ is the (average) valuation function. R determines θ. We can abbreviate $\theta(R, \alpha, \pi)$ to $\theta(\pi)$.

Define θval analogously to ρval.

The error of the (average) valuation (for π, R and α),

$$\operatorname{err}\theta(R, \alpha, \pi) \overline{\text{def}} \begin{cases} \exp\sqrt{\dfrac{1}{\dfrac{1}{(\ln \operatorname{err}\nu(\pi))^2}+\dfrac{1}{(\ln \operatorname{err}\rho(\pi))^2}}}, & \text{if } R \text{ has more than } ||J|| \text{ members,} \\ \operatorname{err}\nu(\pi), & \text{if } R \text{ has between one and } ||J|| \text{ members,} \\ \infty, & \text{if } R = \phi. \end{cases}$$

Subsection 6.1.4 will derive a similar combination of usefulness values, and there it will be shown why the error should be as above.

4.3. ITERATIVE REVISION

In section 2.3 we began a description of the iterative system, detailing the first, or solving step. It was stated that the evaluation function from the previous iteration, together with the training problem set for a current iteration, determine the final state graph set for the current iteration; i.e. $G_I = G(\theta_{I-1}, P_I)$. Although we cannot yet formally describe the details of the creation and alteration of regions, we can define some basic terms and relationships. The second step of an iteration accepts a special set of regions, called the cummulative (region) set, along with the final state graph set, and computes a new cummulative set. More specifically, the cummulative region set from iteration I-1, C_{I-1}, together with the final state graph set, G_I of iteration I determine the cummulative region set, C_I, of iteration I. $C_I = C(C_{I-1}, G_I)$. ($C_0 = \phi$.) This constitutes the second step of iteration I or the region handling step of iteration I. The complex mechanism for the region handling step is the subject of chapters five and six, although an overview is presented shortly.

The third step of iteration I, or regression step of iteration I is the calulation of the valuation function from the cummulative region set for iteration I; so if π is a state, and α a confidence level, the valuation function of iteration I is

$\theta_I(\alpha, \pi) \overline{\overline{\text{def}}} \theta(C_I, \alpha, \pi)$.

The abbreviation $\theta_I = \theta(C_I)$ expresses the relationship in which we usually are interested when considering the iterative processes. We can say that θ_I and C_I are <u>associated</u>.

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

The following fills in some sketchy details of the region handling step in the context of the entire iterative process. Suppose that a feature set, F is given (and space F). Let us examine the first iteration of a series. "CLUSTER" (chapter five) generates the first non-empty set of cummulative regions, C_I. It does this by beginning with the single region $R = (r, g(F, G_1, r), t(F, G_1, r), 1, 0)$, where r is the smallest rectangle which will surround all the points in F corresponding to states in $G_1 = G(\theta_0, P_1)$. CLUSTER tentatively splits the rectangle r into two smaller rectangles. Many such tentative splits of r are examined; and the splits occur in each of the feature 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 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 no further "discrimination" can occur. If the feature set is "useful", then there is eventually more than one element in the resulting region set C_1 .

C_1 becomes the cummulative region set for the regression step of iteration one. The new valuation function, θ_1 becomes something other than the constant 0.5: either ν_1 or else an average of ν_1 and ρ_1 , if the number of regions in C_1 is large enough to warrant the regression, as we discussed in the last section.

This completes the first iteration in the formation/modification of the valuation function. Succeeding steps are a little different. Instead of CLUSTER, directly, another algorithm, "REVISE" is used. Suppose $R \in C_I$. To improve the accuracy of the usefulness estimate, REVISE modifies val(R) (and err(R)) according to the elementary usefulness for G_I, not just of r(R), but of all rectangles in C_I . The elementary usefulness 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 an absolute probability, but rather a conditional usefulness.

REVISE then proceeds to refine each updated region, 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 situation, however, is more complex during later iterations when θ_1 is not a constant. In addition to the inherent difficulty surrounding the conditionality of new elementary usefulness values, there is a systematic bias in the elementary usefulness, which should be taken into account when two subregions are split.[1] REVISE is the subject of discussion in chapter six.

[1] We could write $C_I = C(C_{I-1}, \theta_{I-1}, G_I)$ instead of $C(C_{I-1}, G_I)$ but θ_{I-1} is determined by C_{I-1}.

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 of subrectangles of R. The clustering algorithm which is used is in some ways similar to some well known algorithms; splitting is incorporated [Hartigan]. In other respects, however, "CLUSTER" is more unusual. The distance is non-metric, and it is a property of the "experience" of the system; specifically, through the count functions. 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

Suppose we are given two unreduced regions, $R_1 = (r_1, \gamma_1, \tau_1, \kappa_1, \epsilon_1)$, and $R_2 = (r_2, \gamma_2, \tau_2, \kappa_2, \epsilon_2)$, and suppose that $val(R_1) \geq val(R_2)$. We define the *distance between* R_1 *and* R_2 to be

$$\text{dist}(R_1,R_2) \overline{\overline{\text{def}}} \begin{cases} -\infty, & \text{if } \gamma_1 = 0 \text{ or } \tau_i < 20 \ (i=1,2) \\ \log[\text{val}(R_1)/\text{val}(R_2)] + \text{bias} \\ \quad - [\text{lnerr}(R_1) + \text{lnerr}(R_2) + \text{biaserr}] & \\ & \text{otherwise.} \end{cases}$$

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

Bias and biaserr are two terms which will not be fully defined until chapter six. For iteration one, however, and in fact whenever the valuation function, θ_I does not include a regression component, bias and biaserr are both exactly zero.

If $\text{dist}(R_1,R_2) < 0$, we say that R_1 and R_2 are <u>similar</u>. If R_1, R_2, R_3, R_4 are regions, and if $\text{dist}(R_1,R_2) > 0$, $\text{dist}(R_3,R_4) > 0$, and $\text{dist}(R_1,R_2) > \text{dist}(R_3,R_4)$ then R_1 is "farther" from R_2 than R_3 is from R_4, or the first pair is more <u>dissimilar</u> than the second. Obviously, the larger the combined deviations of R_1 and R_2, and the larger the confidence factor (the product of these forms the error terms), 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.

We can make a comparison here with confidence intervals. Let $x_1 = \log(\text{val}(R_1))$ and $x_2 = \log(\text{val}(R_2))$, and estimate the standard deviation of x_1 by $s_1 = \text{lndev}(R_1)$, that of x_2 by $s_2 = \text{lndev}(R_2)$. Using the Student t distribution, we obtain the confidence intervals:

$$x_1 - t_{\alpha/2}\frac{s_1}{\sqrt{\tau_1}} < \mu < x_1 + t_{\alpha/2}\frac{s_1}{\sqrt{\tau_1}}, \quad \text{and}$$

$$x_2 - t_{\alpha/2}\frac{s_1}{\sqrt{\tau_2}} < \mu < x_2 + t_{\alpha/2}\frac{s_2}{\sqrt{\tau_2}},$$

where μ is the true mean of both x_1 and x_2, assuming they were selected from the same population, and $t_{\alpha/2}$ has the usual meaning. So if $x_1 > x_2$

$$x_1 - t_{\alpha/2}\frac{s_1}{\sqrt{\tau_1}} < x_2 + t_{\alpha/2}\frac{s_2}{\sqrt{\tau_2}}, \quad \text{or}$$

$$(x_1 - x_2) - t_{\alpha/2}(s_1/\sqrt{\tau_1} + s_2/\sqrt{\tau_2}) < 0.$$

Now, we can make a rough estimation of $t_{\alpha/2}$ as a constant. And we replace τ_1 and τ_2 by unity, since the selection of states which composed R_1 and R_2 was not random, so a large total count does not cause a corresponding reduction in the error. The inequality then becomes $(x_1 - x_2) - c(s_1 + s_2) < 0$. The left side is now the distance $dist(R_1, R_2)$ (without the bias terms), with c the confidence factor.

If we were to consider a number of pairs of regions, and if the assumptions made above affect the estimates for each region fairly uniformly, then we would chose the most dissimilar pair (if one exixts) to be the pair most likely to represent two distinct populations. We shall use this reasoning in the clustering algorithm, which we are now equipped to define.

5.2. THE CLUSTERING ALGORITHM

Suppose we are given a feature space F defined by the set, $F = \{f_1, f_2, \ldots, f_m\}$; 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 ith basis vector (for feature f_i), and maxspansteps is a parameter which is a positive integer. As the algorithm implicitly shows, maxspansteps influences the Euclidean feature space distances between successive tentatively inserted boundaries.

CLUSTER(F , G, R_0)

```
    R := { R0 } ;
loop: select some R ∈R which has not been marked;
      comment find best boundary;
      a := lp(r(R));   b := up(r(R));
      bestdist := - ∞ ;
      for i:=1 until n do;
        begin
        length := (b-a) . vi;
        span := entier(length/maxspansteps) + 1;
        for k:=1 step span until length-span do;
          begin
          lp(r1 ) := a;   up(r1 ) := a + k*span*vi;
          lp(r2 ) := a + (k+1)*span*vi;   up(r2) := b;
```

```
        R1 := (r1, g(G,r1), t(G,r1), 1, 1);
        R2 := (r2, g(G,r2), t(G,r2), 1, 1);
        distance := dist(R1,R2);
        if distance > bestdist then r' := r1
        end
      end;
    comment positive distance means regions dissimilar so
      split permanently;
    if bestdist>0 then do;
      begin
      R  :=  R - { R} ;
      R1 := (r', g(G,r'), t(G,r'), 1, 1);
      R2 := (r(R)-r', g(G,r(R)-r'), t(G,r(R)-r'), 1, 1);
      R  := R  ∪  { R1,R2}
      end;
    else mark R;
    if ∃ some unmarked R∈R  then go to start;
```

We call the partitioned set, CLUSTER(F, G, R_0) the output set of CLUSTER (for parent region R, feature set F and final state graph set G). If the output set has more than one member, we can say that the feature set F is useful (within R_0 for G). Similarly the particular features of F in whose dimensions splitting occurred, are useful. Useful features discriminate (among states) (according to G). If a feature space is understood, we can abbreviate CLUSTER(F, G, R_0) to CLUSTER(G, R_0).

Example 5.2

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

The final state graph set, G_1 resulted from a breadth-first search, and 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 (maxspansteps= ∞), which are given in table 5.2.

Table 5.2 **CLUSTER** Output

i	R	$\underline{lp}(r_i)$	$\underline{up}(r_i)$
1	$(r_1, 20, 49, 1, 1)$	(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)

5.3. COMBINATORIAL PROPERTIES OF THE ALGORITHM

Note that the procedure always halts, because the training problem set and thus the counts are finite.

Suppose that a feature set F defines a space F, that G is a final state graph set, and that R is a parent re-

gion, these constituting input for CLUSTER. There are three factors affecting the speed of CLUSTER. One is the number of distinct points of F contained in R, which correspond to states in G. The computer implementation of the count functions actually maps states in G to point regions only once and thereafter the system ascertains whether these point regions are enclosed by subregions of the parent region, R. Thus, only points in F 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. In any case, the time for calculation of the count pair for each split increases no faster than $\tau(G,R)$. If R becomes permanently split, into R_1 and R_2, then the counts for subregions of R_1 and R_2 are still smaller, since $\tau(G,R_i) < \tau(G,R)$ $(i=1,2)$. So, as regions split permanently, the time decreases for counting in an individual region.

The second factor affecting the speed is the total number of possible tentative boundaries that can be inserted in a rectangle. Let us consider the case for which maxspansteps= ∞. In dimension i, the number of boundaries is $up_i(r(R)) - lp_i(r(R))$, so the total number for all dimensions is

$$\sum_{i=1}^{n} [up_i(r(R))-lp_i(r(R))] = [\underline{up}(r(R))-\underline{lp}(r(R))] \cdot \underline{v},$$

where $\underline{v}=(1,1,\ldots,1)$.

The third factor governing speed is the eventual number of permanent splits, k. In practice this number has generally been small, never more than a few.

Thus the total time that CLUSTER requires is less than $k[\tau(G,R)][\underline{up}(r(R))-\underline{lp}(r(R))].\underline{v}$. The most important fact is that the number of features can easily be increased; if each feature has roughly the same range of values, the time increases only linearly. (If the feature ranges are huge, maxspansteps can limit the number of boundaries in appropriate dimensions, with little practical consequence.)

5.4. SHRINKING REGIONS

Suppose that $F = \{f_1, f_2, \ldots, f_n\}$ is the relevant feature set, and R is an output set of regions computed by CLUSTER, for F and the input graph set, G. We define an algorithm, SHRINK, which reduces the sizes of rectangles of regions to enclose minimally the feature space points corresponding to G, as long as the shrinking does not leave gaps between rectangles. (Just the peripheral boundaries are affected.)

SHRINK(G , R)

```
for j=1 until m do;
  comment find rectangle, r, which minimally encloses all
    points in R ;
  begin
  S := { f(π) ∈ R_j | π ∈G ∈ G}  = { p_1, p_2, ... , p_q} ;
  lp(r) := ∞ ;   up(r) := -∞;
  for k:=1 until q do;
    begin
    lp(r) := min(lp(r), p_K);[1]
    up(r) := max(up(r), p_K) [1]
    end;
  comment reduce R_j to r in dimension i iff no gaps result;
  for i:=1 until n do;
    lp(r') := lp(r(R_j)) - v_i ;   up(r') := up(r(R_j));
    for k:=1 until j-1, j+1 until m do;
      if r ∩ r(R_K) ≠ φ  then go to L;
    lp_i(R_j) := lp_i(r);
    L: (similar statements for upper boundary points)
    end
  end;
comment {R_1, R_2, ... , R_m} is output region set;
```

The resulting set, SHRINK(G, R) is the output set of SHRINK (for G and R).

[1] min and max are vector functions which select the minimum and maximum corresponding elements.

5.5. THE FIRST ITERATION OF A SERIES

We are now in a position to define exactly what the evaluation is after the end of the first iteration. In chapter four we stated that the initial valuation function, θ_0, is always a constant, 0.5. Let P_1 be the training problem set for (step one of) the first iteration. We determine the final state graph for the first iteration, $G_1 = G(\theta_0, P_1)$. Suppose that the relevant feature set is $F = \{f_1, f_2, \ldots, f_n\}$. Let π represent a state in $G \in G_1$. Find the rectangle, r, which minimally surrounds all $\underline{f}(\pi)$, for all $\pi \in G \in G_1$. Set R := (r, g(F, G_1, r), t(F, G_1, r), 1, 1); R becomes the parent region for the clustering algorithm, as we calculate:

R := CLUSTER(G_1, R);

R := SHRINK(G_1, R);

C_1 := { red(R) | R $\in$ R};

The set C_1 is the (reduced) cummulative region set of iteration one. We can denote this $C_1 = C(\theta_0, G_1)$ where C represents the above algorithm (for a first iteration only; post-initial iterations require the complex procedure detailed in chapter six).

Finally, we have the valuation function of the first iteration, $\theta_1 := \theta(C_1)$ (see section 4.2).

Example 5.5

The parent region, $R_0 = (r_0, 20, 2721, 1, 1)$ of example 5.2 was the smallest rectangle which could enclose all feature space points 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 5.5 shows the effect of SHRINK.

Table 5.5 SHRINK Output

i	R	$\underline{lp}(r_i)$	$\underline{up}(r_i)$
1	$(r_1, 20, 49, 1, 1)$	(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)

In this case, the valuation function, θ_1 is identical to the region valuation function, ν_1, because the number of cummulative regions is not yet great enough to warrant a regression.

6. REVISING PROBABILITY ESTIMATES AND REFINING REGIONS

Once a non-trivial valuation function has been used in a solution step (i.e. after iteration one), the situation becomes more complex. For example the corresponding count ratios (elementary usefulness values) no longer reflect absolute probabilities, and so cannot be used directly to improve usefulness estimates for cummulative regions. It is possible to revise the values indirectly, however.

In addition to the updating of established regions, this chapter describes how these old regions can be further subdivided using CLUSTER and the new data.

Also, sometimes the established region set does not accommodate all of the feature space maps of the recent final state graphs, so the old regions are extended to surround all new feature space points. After this, another round of possible splitting takes place.

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) usefulness revision of established regions, (2) subdivision, and (3) extension, and subdivision of extensions.

First let us consider how we might revise probability estimates of old regions.

6.1. REVISING USEFULNESS OF ESTABLISHED REGIONS

6.1.1. OVERVIEW OF THE SITUATION AND RELATIONSHIPS

Consider what happens during the solving step of a second iteration, when for the first time a non-trivial valuation function guides the search.

Suppose that a cummulative region set, C_1 has been used to construct a valuation function, θ_1 and that θ_1 has determined a final state graph set, G_2. (For simplicity, assume that the regession component of θ_1 is nil.) One might suspect that a possibility exists of utilizing the information in G_2 to update C_1 and thus improve θ_1. The situation is not simple, however, because the non-trivial search strategy governed by θ_1 implies a very non-random state collection in G_2. More specifically, consider the following.

Assume that R and Q are two regions in C_1, with val(Q)>val(R), and imagine the process of explicit state formation in a solution attempt. Suppose that some state, π, is being developed, and that $\underline{f}(\pi) \in r(R)$. Suppose also that it occurs that one of the immediate offspring of π, π', falls into a "better" area of the feature space, e.g. that $\underline{f}(\pi') \in r(Q)$. Then the strategy will "grab" π' for development, and perhaps thereafter most or all of the explicitly created states succeeding π' will have a usefulness of val(Q) or greater, so that R may seldom or never

again participate. This can mean that, of all the states which eventually map into R, a relatively high proportion lie on a solution path. This in turn implies that the elementary usefulness of r(R) for G_2 is considerably greater than the established usefulness, i.e. that $u(G_2, r(R)) = g(G_2, r(R))/t(G_2, r(R)) >> val(R)$.

This is also reasonable from another, slightly different point of view. Val(R) supposedly represents an absolute probability whereas $u(G_2, r(R))$ reflects a probability which is conditional on the search strategy for G_2. If $u(G_2, r(R))$ were equal to or less than val(R), the number of states developed in order to find one used in a solution would be equal to or greater than the number if a breadth-first search had been in effect; and if this happened generally harder problem instances could not be solved, and the heuristic would be useless. On the other hand, the closer the conditional (elementary) usefulness values are to unity, the better the strategy is working, since there are then few states developed that are wasted.

So we have a situation in which we can calulate counts for G_2, and thus find values for elementary usefulness, but we know that generally, they do not indicate absolute probabilities. Thus we cannot improve the estimates carried by the established cummulative regions simply by averaging those estimates with the corresponding elementary values for G_2.

In addition to this phenomenon of bias or conditionality, there is a further complication. Suppose, for example, that val(R)=0.01 and val(Q)=0.2. Then $u(G_2, r(R))$ might well be considerably greater than 0.01, but $u(G_2, r(Q))$ cannot possibly be larger than 0.2 by a factor of more than five. There is not a simple proportional relationship between the absolute and conditional probabilities.

On the other hand, if val(R) = val(Q) = 0.01, with $u(G_2, r(R)) = 0.2$ and $u(G_2, r(Q)) = 0.05$, then we would suspect that val(R) is too low and/or val(Q) too high. There does exist useful information in the current elementary usefulness estimates, and we can develop a way to extract this information. We shall use all the elementary values to support and smooth each other (with established absolute values as a basis), and to estimate an appropriate multiplier for each which will convert the conditional value to an estimated absolute probability.

Let $\mathcal{R}$ be a cummulative region set, and $\mathcal{G}$ the appropriate final state graph set. Suppose that $R \in \mathcal{R}$ and R' is a new, corresponding "immediate" region with the same rectangle as R, but with $\kappa(R')=1$ (tentatively) and $\gamma(R')=g(G, r(R))$ and $\tau(R')=t(G, r(R))$. We essentially shall postulate that the modified multiplier of R', $\kappa(R') = \psi[val(R)/u(G, r(R))]$, where $R \in \mathcal{R}$ and ψ is a function whose form is fixed for all cases, and use a regression. We

shall choose ψ in such a way as to make use of all of the established cummulative regions and also all the elementary usefulness values of their rectangles for G. Then we shall be able to improve the estimate val(R) by averaging with val(R') = κ(R').u(G,r(R)).

6.1.2. FORMALLY RELATING IMMEDIATE TO CUMMULATIVE REGIONS

Suppose we have a cummulative region set, R over a feature set, F, and a resulting valuation function, θ for R, which has in turn determined a final state graph set, G. Define the (unmodified) immediate region set (for R) (over G),

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

If $R \in R$, $R' \in I$, and r(R)=r(R'), we wish to find a good estimate of the true usefulness, val(R'), i.e. to alter κ(R') so that val(R') = g(G,r(R))/t(G,r(R)). κ(R'). Let $R = \{R_1, R_2, \ldots, R_m\}$ be a cummulative region set, θ a valuation function, G the appropriate final state graph set, and $I = I(R,G)$ the immediate region set over G for R. Let us hypothesize that

$$\log(\text{val}(R')) = b.\log(\theta\text{val}(R)) \qquad 6.1.2$$

where $R \in R$, $R' \in I$, and b is a constant. (This has the property that θval(R)=1 implies val(R')=1 and has a general rough agreement with the observed relationship.)

We can abbreviate $\log(\theta \text{val}(R_i))$ to x_i and $\log(\text{val}(R_i'))$ to y_i, and we have a regression curve $y=bx$. Now, the reliability of each datum varies, so we shall weight the values. Let the variance-covariance matrix be given by

$$V\sigma^2 = \begin{bmatrix} s_1^2 & & & \\ & s_2^2 & & \\ & & \ddots & \\ & & & s_m^2 \end{bmatrix}$$

where s_i is a (logarithmic) standard deviation estimate for R_i' which we shall detail shortly. Then $b = (\underline{X}' V^{-1} \underline{X})^{-1} X' V^{-1} \underline{Y}$, where $\underline{X}$ and $\underline{Y}$ are the m-dimensional vectors composed of the x_i and y_i.[1] This becomes

$$b = \frac{\sum_{i=1}^{m} \frac{x_i y_i}{s_i^2}}{\sum_{i=1}^{m} \frac{x_i^2}{s_i^2}}.$$

The variance of b, $V(b) = (\underline{X}' V^{-1} \underline{X})^{-1}\sigma^2$ and $V(y) = x^2 V(b)$.[1] So the estimated V(y)

$$= x^2 \frac{\sum_{i=1}^{m} \frac{(y_i - bx_i)^2}{s_i^2}}{\sum_{i=1}^{m} \frac{x_i^2}{s_i^2}}.$$

Now, $x_i = \log(\theta \text{val}(R_i))$ and $y_i = \log(\text{val}(R_i'))$ and not only y_i but also x_i has an associated error. The

[1] See [Draper & Smith].

logarithmic deviation in y_i is $\mathrm{lndev}(R_i')$ (section 4.2) and the (logarithmic) deviation in x_i is $\mathrm{lndev}\theta\mathrm{val}(R_i)$ (section 4.3). It is equivalent to assume no error in x_i and to translate the actual x_i error to a component of the y_i error. Thus we have $s_i = \mathrm{lndev}(R_i') + b.\mathrm{lndev}\theta\mathrm{val}(R_i)$. Since b is not known initially, the algorithm for its calulation, given in the next subsection, will be iterative.

6.1.3. SETTING MULTIPLIER VALUES OF IMMEDIATE REGIONS: KMOD

Now we are in a position to define the algorithm, KMOD, which computes a multiplier $\kappa(R_i)$ for each region R_i. It initiates this process by performing the regression just described. After the value of b is discovered, each $\kappa(R_i)$ is appropriately modified.

Let θ, R, G be as above, again with $I = I(R,G)$ the immediate region set over G for R. Below is the algorithm which sets the multipliers of I; it is further explained in the text following.

KMOD(θ, R, G)

```
comment R = { R1, R2, ... , Rm }  &  I = { R1', R2', ... , Rm' } ;
for i:=1 to m do;
   begin
   b:=1;
1: lastb := b ;
```

s_i := $\text{lndev}(R'_i) + b*\text{lndev}_\theta\text{val}(R_i)$;

$b := \sum_{i=1}^{m} (x_i y_i / s_i^2) \;/\; \sum_{i=1}^{m} (x_i^2 / s_i^2)$;

$\text{bvar} := \sum_{i=1}^{m} [(y_i - bx_i)^2 / s_i^2] \;/\; \sum_{i=1}^{m} (x_i^2 / s_i^2)$;

$\kappa(R'_i) := \exp(x_i - b*x_i)$;

$\epsilon(R'_i) := \exp\left[c*(\sqrt{x^2 * \text{bvar}} - b*x_i) + \text{lnerru}(R_i)\right]$;

if b-lastb > 0.1 then go to 1

end;

The resulting set of regions, $M = M(R, I) = \text{KMØD}(\theta, R, I)$ is called the modified immediate (region) set (of I) (for R).[2] Notice that $b = b(R, I)$. We label b the log-usefulness factor relating I to R.

Suppose $R \in \mathcal{R}$, $R' \in I$ and $R'' \in M$, with $r(R) = r(R') = r(R'')$ (i.e. R is a cummulative region, R' a corresponding unmodified immediate region carrying elementary usefulness, and R" a corresponding modified immediate region). According to KMØD, the multiplier of R" is $\kappa(R'') = \exp(x-bx) = \exp[\log(\theta\text{val}(R)) - b*\log(\theta\text{val}(R))]$ which is $\theta\text{val}(R)$ divided by the predicted (elementary) usefulness of $r(R')$. Thus $\text{val}(R'') = \kappa(R'').u(G, r(R)) = \text{val}(R).u(G, r(R))$/predicted elementary usefulness of $r(R) \doteqdot \text{val}(R)$. The accuracy of the approximation is governed by the correctness of the regression.

[1] Recall that c, the confidence factor converts logarithmic deviations to logarithmic errors.

[2] Recall that the cummulative region set determines the valuation function.

6.1.4. COMBINING CUMMULATIVE AND IMMEDIATE REGIONS

Now it is a comparitively simple matter to use the modified immediate regions to update the corresponding cummulative ones. Since the reliability is different for each of the two members of a corresponding pair, we shall weight the usefulness values accordingly.

It can be shown that the variance of a weighted average of normally distributed variables is a minimum if the weights are the inverses of the original variances. Since we have assumed a log-normal distribution for the usefulness, we have the following.

Let $R = \{R_1, R_2, \ldots, R_k\}$ be a set of regions. The *average usefulness of the regions of* R,

$$\underline{\text{avgval}}(R) \underset{\text{def}}{=\!=} \exp\left[\frac{\sum_{i=1}^{k} \frac{\log(\text{val}(R_i))}{\text{lndev}^2(R_i)}}{\sum_{i=1}^{k} \frac{1}{\text{lndev}^2(R_i)}}\right].$$

Computation gives the variance of $\log(\text{avgval}(R))$ as $k \Big/ \sum_{i=1}^{k} [1/\text{lndev}^2(R_i)]$. However, the sets of regions whose usefulness values will be averaged will be considered to be samples of a single population, so the law of large numbers reduces the numerator of this variance expression to unity. And we have the *deviation of the average of* R,

$$\underline{\text{devav}}(R) \underset{\text{def}}{=\!=} \exp \sqrt{1 \Big/ \sum_{i=1}^{k} \frac{1}{\text{indev}^2(R_i)}} \quad .$$

6.1.5. THE USEFULNESS REVISION SUBSTEP

Finally, we are in a position to sum up precisely how immediate regions are used to revise the usefulness estimates of the cummulative set. Suppose that a cummulative set and its valuation function are R and θ, respectively, and let G be the final state graph for θ. The usefulness revision algorithm is given below.

USFREV(R , θ , G)

Create I, the unmodified region set for R over G ;

M := KMOD(θ , R , I);

Q := { red(R) | $R \in M$} ;

U := { T | r(R) = r(Q), $R \in R$, $Q \in Q$ and

T = (r(R), avgval({ R,Q}), errav({ R,Q }))} ;

The set U = USFREV(R, θ , G) is the usefulness-revised (region) set of R (for θ and G). If R is C_{I-1}, the cummulative set of iteration I-1, θ is θ_{I-1}, the valuation function of iteration I-1, and G is G_I, the final state graph set of iteration I, then USFREV(θ_{I-1}, C_{I-1}, G_I) is the usefulness-revised (region) set of iteration I. A side

effect of KMOD is the calculation of $b_I = b(C_{I-1}, I_I)$; this log-usefulness factor of iteration I will be used later on.

6.1.6. AN INTERPRETATION OF THE LOG-USEFULNESS FACTOR

From equation 6.1.2 we have b = log(val(R') / log(θ val(R)) , where R is from the cummulative region set, and R' from the unmodified immediate set. We could think of this as being roughly equivalent to b = log(usefulness probability in strategy) / log(absolute usefulness). So the lower the value of b, the better the heuristic is working. We can define the power of a valuation function (for the relevant training problem set) as 1/b - 1.

Example 6.1

The following is a continuation of our standard fifteen puzzle example. The cummulative set, C_1 was $\{(r_1, 0.25, 1.8), (r_2, 0.016, 6.2), (r_3, 0.00019, 25.)\}$. The training problem set for the second iteration consisted of twelve puzzle instances whose depths (moves from goal) ranged between twelve and fourteen. The unmodified immediate region set was $I_2 = \{(r_1, 67, 197, 1, 1), (r_2, 19, 106, 1, 1), (r_3, 27, 1783, 1, 1)\}$. Applying KMOD gave the modified immediate set $M_2 = \{(r_1, 67, 197, 0.54, 2.6), (r_2, 19, 106, 0.16, 18.),$

(r_3, 27, 1783, 0.022, 420)} . The log-usefulness factor, b_2 was 0.55. Reducing the modified set, we obtain: Q_2 = {(r_1, 0.18, 3.1), (r_2, 0.028, 25.), (r_3, 0.0033, 510)} . And merging Q_2 with C_1, via AVGVAL, we have the usefulness-revised set, U_2 = {(r_1, 0.24, 1.7), (r_2, 0.019, 4.8), (r_3, 0.00021, 18.)} . (Notice the magnitudes of the errors are reduced.)

Since b=0.55, the power is 0.8. The usefulness of the second revised cummulative region, above, was 0.19 = 1/53. We would expect the search strategy of the first iteration to have made use of exp(0.55*log(0.019)) = 0.11 of the states mapping to that region, or about one in 9, as opposed to one in 53 for a breadth-first search. Comparing this figure with the corresponding region of the unmodified immediate set, we find that about one in 6 was actually used.

6.2. SUBDIVIDING ESTABLISHED REGIONS

After the established cummululative regions are modified as decribed in the preceding section, each region of that resulting usefulness-revised set is possibly subdivided in a manner basically similar to the clustering which 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 will

be called once for each region of the established cummulative set, with that region taking the role of the parent region. The situation is a little more complex now, however, and we shall need need two of the formalisms developed in the last section, as well as another algorithm to adjust multipliers.

6.2.1. ALGORITHMS SUBDIV AND KALTER

Suppose a final graph set , G has been determined by a valuation function θ, and that θ is associated with a cummulative region set R. Suppose, also that $U = \{R_1, R_2, \ldots, R_m\}$ is the resulting usefulness-revised set of R, b an appropriate log-usefulness factor (see later), and bvar its estimated variance. The second substep in the revision of R is a subdivision of the regions of U, and is defined by the algorithm:

SUBDIV(θ, G, b, bvar, U)

comment $U = \{R_1, R_2, \ldots, R_m\}$;
for i:=1 to m do;
  begin
  T_i := CLUSTER(G, R_i);
  KALTER(θ, b, bvar, R_i, T_i)
  end;
$T := \bigcup_{i=1}^{m} T_i$;

$S := \{ S \mid T \in T \ \& \ S = red(T) \}$;

S = SUBDIV(θ, G, b, bvar, U) is the subdivided (region) set of U (for θ, G, b, and bvar).

The algorithm, KALTER, modifies the multiplier values for each subregion in T_i. It takes into account two phenomena, both of which basically have already been discussed in section 6.1. The first is that immediate regions are conditional on their search strategy, and generally must have their multipliers reduced in order to reflect absolute usefulness. This is easily accomplished since the parent region, R_i embodies absolute usefulness, so the multiplier value for each of the new subregions, T_j, of R_i is just the usefulness of R_i divided by the average elementary usefulness of the T_j.

The second phenomenon relates to the relationship between immediate and cummulative regions through the log-usefulness factor, b. Within the boundaries of a cummulative region, the (regression component of the) valuation function has caused the search strategy to favour the "good" corners of the rectangle, and this has supposedly biased the poorer elementary usefulness values, upward. In subsection 6.1.3 we made appropriate adjustment for this bias over the set of cummulative regions; now we must adjust biased elementary usefulness values, within a cummulative region, in order to find reasonable absolute usefulness estimates for its new subregions.

If T_1 and T_2 are two subregions, the difference of the logarithms of their predicted usefulness is $\log(\theta\,val(T_1)) - \log(\theta\,val(T_2))$.[1] According to equation 6.1.2 this translates to a predicted difference in logarithmic elementary usefulness of $b\,[\log(\theta\,val(T_1)) - \log(\theta\,val(T_2))]$. The remainder of the absolute difference is hidden in the bias. This remainder or bias is $[1-b]\,[\log(\theta\,val(T_1)) - \log(\theta\,val(T_2))]$.

In the following formalization of KALTER, let be a valuation function, b a suitable log-usefulness factor and bvar 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.

KALTER(θ, b, bvar, R, T)

```
comment T = {T1, T2, ... , Tk} ;
comment first convert conditional usefulness of immediate
        set to absolute usefulness;
for i:=1 until k do;
   begin
   κ(Ti) := κ(Ti) * val(R)/avgval(T) ;
   ϵ(Ti) :=  [ϵ(Ti)] [err(R)] [errav(T)]
   end;

comment correct for any bias;
```

[1] Note that the regression component, ρ, alone, without ν, could be substituted for θ, since R_1 and R_2 are within a single region of the former valuation function.

```
for i:=1 until k do;
   begin
   bias  :=   [1-b] [ log( θ val(T_i))-log( θ val(R))] ;
   biassdv :=  √bvar * [ log(θ val(T_i))-log(θval(R))]^2   ;
   κ (T_i) := exp(bias)* κ(T_i) ;
   ∈ (T_i)
       := exp [ ln∈(T_i)+lnerrθval(T_i )+lnerrθval(R)+biassdv*c]
   end;
```

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 biaserr, were zero unless the regression component of the valuation function was non-trivial. This bias is exactly what was introduced in the preceding subsection. So we can now fully define the distance.

Suppose that R_1 and R_2 are two regions with $val(R_1) \geq val(R_2)$, and let b_I be the log-usefulness factor of iteration I. Also, let θ_{I-1} be the valuation function of iteration I-1. Then the distance between R_1 and R_2 for iteration I is

$$\text{dist}(R_1, R_2, \theta_{I-1}, b_I, \text{bvar}_I) \overline{\text{def}} \begin{cases} -\infty, & \text{if } \gamma(R_1)=0 \text{ or } \tau(R_i)<20 \ (i=1,2) \\ \log[\text{val}(R_1)/\text{val}(R_2)] + \text{bias}_I & \\ \quad - [\text{lnerr}(R_1) + \text{lnerr}(R_2) + \text{biaserr}_I] & \\ & \text{otherwise.} \end{cases}$$

where $\text{bias}_I \overline{\text{def}} [1-b_I][\log[\theta_I \text{ val}(R_1)/\theta_I \text{val}(R_2)]$ and

$\text{biaserr}_I \overline{\text{def}} c\sqrt{\text{bvar}_I} \cdot \log[\theta_I \text{ val}(R_1)/\theta_I \text{val}(R_2)]$.

6.2.3. THE SUBDIVISION SUBSTEP

If U_I is the usefulness-revised region set, G_I is the final state graph set, b_I and bvar_I are the log-usefulness factor and its variance estimate, all of iteration I, and θ_{I-1} is the valuation function of iteration I-1, then S_I = SUBDIV(θ_{I-1}, G_I, b_I, bvar_I, U_I) is the subdivided (region) set of iteration I.

Examples 6.2

The first example is from the second iteration, for which the valuation function had no regression component; so this will illustrate only the first part of KALTER (no bias within established regions). One of the usefulness-revised regions of example 6.1 was R = (r, 0.24, 1.7), 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,

$R_1 = (r_1, 24, 24, 1, 1)$, $\underline{lp}(r_1)=(1,0,0,0)$, $\underline{up}(r_1)=(2,0,0,3)$;

$R_2 = (r_2, 43, 173, 1, 1)$, $\underline{lp}(r_2)=(3,0,0,0)$, $\underline{up}(r_2)=(5,0,0,3)$.

In KALTER, avgval($\{R_1, R_2\}$) = 0.42. So the multiplier correction factor was val(R)/0.42 = 0.57. Errav($\{R_1, R_2\}$) was 1.2, and combining this with err(R) gave a final error estimate of 2.1. The two new subdivided regions became $(r_1, 24, 24, 0.57, 2.1)$ and $(r_2, 43, 173, 0.57, 2.1)$, which, when reduced are $(r_1, 0.57, 3.2)$ and $(r_2, 0.14, 2.6)$.

For an illustration of the second part of KALTER, we need to consider a case in which the regression component of the valuation is non-trivial. This occurs, for instance, in iteration four of our standard example. The region of the usefulness-revised set which is of concern to us now is R = (r, 0.001, 5.), $\underline{lp}(r) = (7,0,0,2)$, $\underline{up}(r) = (10,0,2,5)$. 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 KALTER changed the multipliers to val(R)/avgval($\{R_1, R_2\}$) = 0.001/0.094 = 0.011. This value was further modified by the bias correstion section. We have:

$$\text{bias} = [1-b]\,[\log(\theta\text{val}(R_i)) - \log(\theta\text{val}(R))]$$

$$= [1-b]\,[\log(\rho\text{val}(R_i)) - \log(\rho\text{val}(R))]$$

$$= [1-b]\,[\underline{a}.(1,\underline{cp}(r(R_1))) - \underline{a}.(1,\underline{cp}(r(R)))]$$

$$= [1-b]\,[\underline{a}\ .\ (0,\ \underline{cp}(r(R_1))-\underline{cp}(r(R)))]\ .$$

The only relevant parameter of the regression valuation function was that for the first feature, and it was $a_1 = -0.51$. The value of b was 0.25. Thus, for R_1, bias = $(0.75)(7.5-8.5)a_1 = 0.38$. And for R_2, bias = $(0.75)(9.5-8.5)a_1 = -0.38$. Thus the final values for the multipliers are: $\kappa(R_1) = 0.011*\exp(0.38) = 0.016$ and $\kappa(R_2) = 0.011*\exp(-0.38) = 0.0073$. The errors will not be calualated here, but the final usefulness estimates 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 cummulative regions do not generally cover all the feature space points mapped from a new final state graph set. So some of the outer cummulative regions need to be extended to engulf the new points. After that, further splitting is allowed, similar to that described in the preceeding section.

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

6.3.1. BOUNDARY EXTENSION

Let $S = \{ R_1, R_2, \ldots, R_m \}$ be a set of subdivided regions. Let G be a final state graph set of the same iteration, and F the appropriate set of features determining the vector function $\underline{f}$. Define the set of states, $\Pi = \{\pi \in G \in G \mid \underline{f}(\pi) \notin R \ \forall \ R \in S\}$, and the set of feature space points $S = \{ \underline{p}_i \mid \underline{p}_i = \underline{f}(\pi), \pi \in \Pi\}$, called the *outlying point set for* S *and* G.

The following algorithm extends the regions.

ENCLOS(G , S)

```
Form the outlying point set, S, for S and G.
comment S = { p1,p2, ... ,pk} ;   S = { R1,R2, ... ,Rm} ;
for i:=1 until k do;
  comment find rectangle to extend to cover p ;
  for j:=1 until m do;
    if ||r(Rj)-pi|| > ||r(R1)-p||  (1≤l≤m) then go to nxtr;
    if r(Rj) ∩ r(Re) ≠ φ  (i≤ l≤m, l≠j) then go to nxtr;
    lp(r(Rj)) := min [lp(r(Rj)),pi] ;
    up(r(Rj)) := max[ up(r(Rj)),pi]
    nxtr: end
  end;
```

IF S is a subdivided cummulative region set, and G is a final graph set, then the set, $S' =$ ENCLOS(G, S) is the

[1] $\| \ \|$ represents Euclidean feature space distance.

uncorrected extended (subdivided) region set of S (for G). (Uncorrected because the counts for the outlying points have not yet been incorporated into the extensions.)

6.3.2. MODIFYING USEFULNESS VALUES OF EXTENSIONS

When a subdivided region set has been extended, its old usefulness value may be an inaccurate estimate for the extension. We can use KALTER to attempt to correct this problem.

Let S' be 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': $I = \{ (r(R), g(G, r(R)), t(G, r(R)), 1, 1) \mid R \in S \}$. Define I', similarly, from S'. Let θ be the relevant valuation function, and b and bvar the appropriate log-usefulness factor and its variance estimate (see later). The algorithm which adjusts the usefulness estimates according to these variables is:

COREXT(θ, G, b, bvar, S, S')

comment $S = \{ R_1, R_2, \ldots, R_m \}$ is a subdivided set,
$S' = \{ R'_1, R'_2, \ldots, R'_m \}$ is the uncorrected extended set of S;

Form immediate sets, $I = \{ Q_1, Q_2, \ldots, Q_m \}$, from S,
and $I' = \{ Q'_1, Q'_2, \ldots, Q'_m \}$, from S';

```
for i:=1 until m do;
   begin
   comment correct Q'i ;
   KALTER( θ ,b,bvar,Qi, { Q'i } );
   comment use the multiplier of Q'i to modify R'i ;
   val(R'i) := val(Ri)*κ (Q'i);
   err(R'i) := err(Ri)*err(Q'i)
   end;
comment { R'1,R'2 ... ,R'm} is the output set;
```

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

To see why this algorithm provides a suitable multiplier adjustment, consider the following argument. Let $R \in S$, $R' \in S'$, and $Q \in I$, $Q' \in I'$ be corresponding regions (whose rectangles are all identical). We already have a reasonable estimate for val(R), and we would like a good estimate of val(R'). Val(R') = val(R)[val(R')/val(R)] $\doteq$ val(R) [usefulness of Q', adjusted for bias / usefulness of Q, corrected for bias] , which agrees with the algorithm.

6.3.3. SUBDIVIDING THE EXTENSIONS

Let E be a corrected extended region set and S the unextended (subdivided) counterpart of E, G be a final

state graph set, and θ its corresponding valuation function, with b and bvar having their usual meanings. Then extension subdivision proceeds:

EXTDIV(θ , G , b, bvar, S, E)

Define $V = \{ R' \in E \mid R' \neq R \ \forall \ R \in S \}$;

W = SUBDIV(θ , G , b, bvar, V) $\cup$ (E - V) ;

W = EXTDIV(θ , G , b, bvar, S, E) is the subdivided extended (region) set of E (for S, θ , G, b, and bvar).

6.3.4. THE EXTENSION SUBSTEP

Suppose θ is a valuation function, G is a final state graph set, b, bvar a log-usefulness factor and its estimated variance, S is the subdivided set (of section 6.2). The full extension algorithm is given below.

EXTEND(θ , G , b, bvar, S)

S' := ENCLOS(G , S);

E := COREXT(θ , G , b, bvar, S, S');

X := EXTDIV(θ , G , b, bvar, S, E);

The set X = EXTEND(θ, G , b, bvar, S) is the final (region) set of S (for θ, G, b, and bvar). If θ_{I-1} is the

valuation function of iteration I-1, and G_I, b_I, $bvar_I$, and S_I represent the usual variables, of iteration I, then C_I = EXTEND(θ_{I-1}, G_I, b_I, $bvar_I$, S_I) is the final or cummulative (region) set of iteration I.

Example 6.3

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), lp(r)=(11,0,0,0), up(r)=(21,2,1,0) and the corresponding uncorrected extended region was R' = (r', 0.00001, 40.5), lp(r')=(11,0,0,0), up(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'). So the multiplier for the corrected extension is given by KALTER(θ_2, b_3, $bvar_3$, Q, {Q'}). In this, the multiplier of Q' is altered: $\kappa(Q')$ = val(Q)/avgval({Q'}) = val(Q)/val(Q') = 0.24. (The errors will not be calculated here.) This accounts for the conditional to absolute probability conversion.

The bias correction proceeds: The parameter vector was a = (0.73, -0.72, , -2.4, 0, 0), and the log-usefulness factor was b = 0.34. Thus the

bias = $[1-b]\,\underline{a}\,.\,[0, \underline{cp}(r(Q'))-\underline{cp}(r(Q))]$

$= 0.66(-0.72, -2.4, 0, 0)\,.\,((16,1,0.5,0)-(34.5,1,1,0))$

$= -8.8.$

And the final value for the multiplier is given by $\kappa(Q') = 0.24*\exp(-8.8) = 0.000036$. So the the corrected extended region, R'' has a usefulness $val(R'') = (.000036)(.00001) = 3.6\times10^{-10}$.

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 $Q_2 = (r_2, 0, 1920, 1, 1)$, $\underline{lp}(r_2)=(31,0,0,0)$, $\underline{up}(r_2)=(58,2,2,0)$. KALTER($\theta_2, b_3, bvar_3, R'', \{Q_1,Q_2\}$) gives (intermediately, for i=1,2) $\kappa(Q_i) = val(R'')/avgval(\{Q_1,Q_2\}) = 3.6\times10^{-10}/0.28 = 1.3\times10^{-9}$. 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 $\kappa(Q_1) = 1.3\times10^{-9}*\exp(6.6) = 9.6\times10^{-7}$ and $\kappa(Q_2) = 1.3\times10^{-9}*\exp(-4.8) = 1.1\times10^{-11}$. And $val(Q_1)$ becomes $(129/457)(9.6\times10^{-7}) = 2.7\times10^{-7}$ while $val(Q_2)$ is $(0.5/1920)(1.1\times10^{-11}) = 2.8\times10^{-15}$.

6.4. THE ENTIRE PROCESS OF REVISION

In section 5.5 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 C_{I-1} is the cummulative region set of iteration I-1, θ_{I-1} and G_I the associated valuation function and final state graph set, then C_I, the cummulative (final) region set of iteration I is calculated:

U_I = USFREV(θ_{I-1} , C_{I-1}, G_I); (6.1.5)

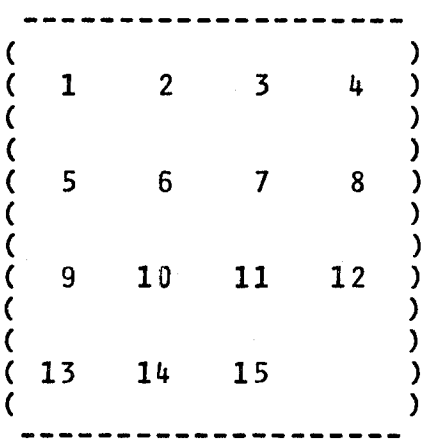
S_I = SUBDIV(θ_{I-1} , G_I , b_I, $bvar_I$, U_I); (6.2.3)

C_I = EXTEND(θ_{I-1} , G_I , b_I, $bvar_I$, S_I); (6.3.4)

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

Appendix A THE STATE-SPACE / FEATURE 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 [Nilsson]. The fifteen puzzle is a common example of such a problem. It is a square which contains fifteen smaller squares or tiles and a space into which adjacent tiles can be slid. The goal might be:



and the starting configuration, or starting state, any of 16!/2 (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).

State-space problems, then, are problems with a precisely defined starting state, an explicit goal state,

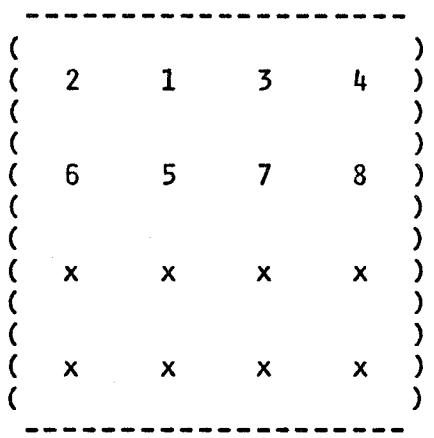
and other distinct states produced as offspring of the starting state by means of a set of operators. The "space" in the term "state-space" refers to the entire set of possible states reachable from any starting state by successive applications of operators. A problem instance is a particular starting state, together with a goal state and set of operators.

The fifteen puzzle might seem like just a toy problem, but it has often been selected as a representative because it is easily formulated but difficult to solve. Many important and hard problems can be formulated as state-space problems, for example theorem proving, and progress with any representative problem is likely to be equivalent to general advancement, as long as our approach is general.

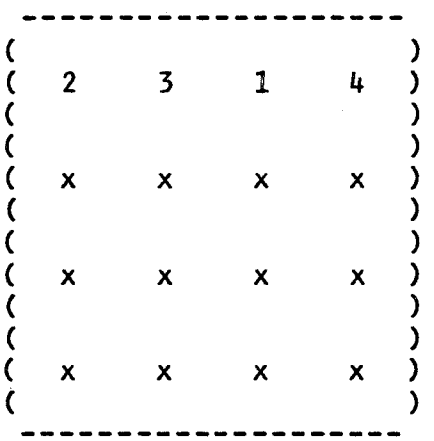
If we consider a starting state as the root of a state tree, then operators correspond to arcs linking nodes (states) to their offspring. (Generally, the tree becomes a graph if each redundant state is considered as a single node.) A graph traverser is a mechanism which (beginning with the starting state) explicitly generates offspring nodes by developing a current node (applying the set of operators). A solution corresponds to a path through the graph which links the starting state to the goal. A graph traverser is guided by some search strategy, the simplest being the breadth-first strategy (i.e. nodes developed in the order of their generation).

A breadth-first search usually uses prohibitive amounts of both space and time, even for simple problems. Growth is exponential. For example, for the fifteen puzzle, the number of states developed at level L (depth of the state tree) is about three to the power L, so practically, a 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 number of possible configurations.) To be effective, a search strategy must lead the graph traverser fairly directly toward the goal, without generating too many extraneous nodes.

One formalism that has been used to define a search strategy is the evaluation function. An evaluation function maps nodes to numbers, so that states can be ranked according to probable "usefulness in a solution" (and the states selected for development in the appropriate order). A simple example of such a function for the fifteen puzzle is the sum, for each tile, of the distance of the tile from its "home" position (ignoring intervening tiles). This does not work very well, because configurations such as



arise. From the point of view of the evaluation function, this appears to be close to the goal, but in fact it most definitely is not. So one might use a more sophisticated function: Σ distances + $c.\Sigma$ reversals [Doran & Michie]. But then states like



occur. The reader may appreciate that, generally, several elementary terms or features might be useful. However, even with just two terms, many separate computer runs have to be made in order to find the best parameter(s), and with more than two features, the cost is prohibitive. (Ordinary

statistical techniques are not appropriate, because random problems are generally too difficult to solve at all.)

In any case, with this state-space paradigm as described to this point in our discussion, there does not seem to be much intelligence apportioned to the computer/program.

The following approach is an attempt to preclude this dilemma by mechanizing a higher level system which deals with feature spaces. Consider a feature space formed by an ordered set of features, each of which defines one dimension. We shall associate an estimated probability of a node's being on a good solution path, or its "usefulness" with an area in the feature space (see also [Michie & Ross]). For ease and conciseness of expression, we can restrict the feature space regions to be rectangular. So if a node in a state space maps into the interior of a rectangle, r_i, in a feature space, it has usefulness u_i; thus the nodes can be ranked for development order. In this system, the usefulness is in fact a combination of two factors: the just mentioned value for the associated rectangle, and also a value obtained by regressing the usefulness figures against the mid points of their associated rectangles.

This usefulness is a product of the system's past experience with problem instances; former results are generalized to apply to future problem instances (tenta-

tively at first).

Notice that we have been talking about three separate spaces. One is the state space of nodes or configurations. Another is the feature space into which the nodes are mapped, and which is partitioned into rectangles with associated usefulness values. And the other is the one dimensional ranking space in which the evaluation function orders nodes.

Next let us consider how these usefulness figures are obtained. When the system begins its operation with some problem such as the fifteen puzzle, it is given an ordered set of features (whose ranges are integer), but the resultant feature space starts as an undifferentiated lump with an undefined usefulness. The system is also provided with a set of problem instances to attempt, at least one of which should be easy enough to solve breadth-first. After all samples have been attempted, the entire set of developed nodes is mapped into the feature space. For each point, a pair of integers, (g,t) is calculated. The "total count", t is the total number of states which map into the point. The "good count", g is the total number of nodes which map to the point, too, but which also lie on a solution path. Thus the ratio, g/t is a measure of the usefulness of the corresponding state. Since these numbers are generally of small magnitude, therefore unreliable, the points, together with their associated integer counts, are clustered. Thus

we obtain (g,t) pairs for rectangles (summing the values for the interior points).

As an example, for the fifteen puzzle, we might define a two-dimensional feature space using the two elementary terms mentioned earlier, the distance-from-home sum and the number of reversals. Then, points close to the origin would be found to have high g/t ratios, and points away from the origin woulgenerally have low usefulness.

The clustering algorithm is a splitting one; and the final number of rectangles is governed by the data. Roughly speaking, the algorithm works this way: A boundary is tentatively inserted to divide a rectangle, r, into two, r_1 and r_2, and the temporary split becomes permanent iff the associated pairs (g_1, t_1) and (g_2, t_2) define "dissimilar enough" usefulness values g_1/t_1 and g_2/t_2, and if that boundary produces the "best" division of all possible choices (an error term is also defined, by the count pairs and otherwise). The process is repeated for each emerging smaller rectangle until no further bisection is sustained by adequately differing (associated) probability estimates.

In our fifteen puzzle, two dimensional feature space example, we would typically find a final set of approximately three rectangles: with usefulness values of about 0.5, 0.02, and 0.001.

For future solution attempts, the system uses this freshly constructed information in the form of an evaluation

function to order and select states. Also, after each solution set attempt, the usefulness values are revised, and the rectangles are further refined (subdivided), using new, current (g,t) counts. Continued experience improves the performance of the dynamic evaluation function.

Appendix B NON-RANDOMNESS ERROR ESTIMATION

In section 3.4 we discussed a non-randomness error which arises because we generalize from easier to more difficult problem instances, and from one search strategy to another. Let us consider just one source of such an error, in a first iteration, when the search is breadth-first. Here, the graph traverser wanders around aimlessly in the feature space, beginning with the starting state. Many of the nodes developed are in fact 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. These poorer nodes are developed indiscriminately, along with the good ones, however, and they end up seeming worse than they really are, in the eyes of the elementary usefulness (the good count being zero). If, on the other hand, the starting state had been farther from the goal, some nodes similar to these "poor" ones may have participated in a solution.

In an attempt to quantify this and other non-randomness error soures, the user-specified error, erru, was selected to be erru = erru(R) = $1/\sqrt{50\text{val}(R)}$, where R is a region. So regions whose usefulness corresponds to a state distant from a solution [1] are considered to have a low reliability.

[1] This is tricky, because a state distant from a solution is perhaps not necessarily one with a low usefulness.

Appendix C USEFULNESS, ERRORS, AND NOTATION

In this paper usefulness values and error terms appear in two forms, logarithmic and non-logarithmic. Because error estimates arise frequently, a standard abbreviation is used throughout: errors are logarithmic whenever they are prefixed by the letters "ln".

In logarithmic form, an upper or lower "likely" limit of the usefulness is obtained by adding or subtracting the appropriate logarithmic error term. In non-logarithmic form, the error becomes a factor or divisor.

Error names usually have the letter sequence "err" as a prefix (and logarithmic errors, "lnerr"). Logarithmic errors are related to logarithmic deviations: lnerr = c.lndev where c is a confidence factor. (The prefixes "dev" and "lndev" distinguish deviations.)

In a couple of instances, e.g. for "biassdv", the errors are encountered in logarithmic form only, and the standard notation is avoided.

A list of errors and deviations and references to their first appearance in the text are given in the index of definitions.

There are several functions mapping states to usefulness values, indirectly, through their feature space locations. Corresponding functions which map (center points of rectangles of) regions in the feature space to usefulness values in the ranking space have the suffix "val".

INDEX OF DEFINITIONS

α (confidence level) 35
associated
  graph & problem sets, and valuation function 14
  cummulative region set & valuation function 39
augmented vector (of feature space point) 34
augmenting function (h) 34
avgval (average usefulness of region set) 61
average usefulness (of a set of regions) (avgval) 61
average usefulness (of a state) 36

b (log-usefulness factor) 57,60
boundary extension 72

c (confidence factor) 26,88
center point (of rectangle) (cp) 34
CLUSTER (clustering algorithm) 45
confidence factor (c) 26,88
confidence level (α) 35
COREXT (counts points wihin extended regions) 73
corner point (of rectangle) (lp & up) 18
corrected extended region set 74
count functions (g and t) 19
counts 19
  good --- 19
  total --- 19
cummulative region set 38
  --- of iteration one 51

determine(s)
  problem set & valuation function --- graph 14
  region set --- valuation function 37
  region set & state graph set --- region set 38
devav (deviation of average of region set) 61-62
devc (count deviation) 24
deviation (error without confidence factor) 88
  count --- (devc) 24
  average --- of region set (devav) 61
  regression --- (devϱ) 35-36
  user-defined --- (devu) 27,87
devu (user-defined deviation) 25
discriminate 46
dissimilar (regions) 43
dist (distance function) 43,69
distance (between regions) 42,68

elementary usefulness (u) 20
ENCLOS (extends regions to cover new points) 72
ϵ (multiplier error of unreduced region) 28
err (total error of a reduced region) 29
err (total error of an unreduced region) 30
err ν (error of region valuation) 32

err θ (error of average valuation) 37
error (see also deviation) 88
--- of average valuation (err$_\theta$) 37
--- of region valuation (err$_\nu$) 32
evaluation function 31
EXTDIV (subdivides extended regions) 75
extended region set 74

feature space (F) 12,17
feature space map (of a state) ($\underline{f}$) 13
final region set 75
final state graph (G) 13,14
final state graph set (G) 14
for iteration I (G_I) 15

g (good count function) 19
γ (good count of unreduced region) 28
good count 19

h (augmenting function) 34

immediate region (set)
modified --- 60
unmodified --- 57
iteration
first --- of a series 51

KALTER (adjusts multiplier of subdivided regions) 67
(multiplier of unreduced region) 28
KMOD (relates immediate to established regions) 59

log-usefulness factor (b) 60
--- of iteration I (b_I) 63
lower point (of a rectangle) ($\underline{lp}$) 17
$\underline{lp}$ (lower point) 17

modified immediate region (set) 60

ν (region valuation function) 32

outlying point set 72
output set
--- of CLUSTER 46
--- of SHRINK 50

parent region (for CLUSTER) 45
power 63
predicted usefulness (of a region) (ρval) 35
predicted usefulness (of a state) 35
problem instance (P) 12

r (rectangle field of region) 28
rectangle 18
center point of --- ($\underline{cp}$) 34

corner point of --- (lp & up) 18
red (map from unreduced to reduced region) 30
region 28
corrected extended --- (set) 74
cummulative --- (set) 38
--- of iteration I 76
--- of iteration one 51
extended --- (set) 74
final --- (set) 75
immediate --- (set)
modified --- 60
unmodified --- 57
modified immediate --- (set) 60
parent --- (for CLUSTER) 45
reduced --- 30
rectangle of --- (r) 30
total error of --- (err) 30
usefulness of --- (val) 30
reduced --- of an unreduced --- (red) 30
subdivided --- set 66
--- of iteration I 69
uncorrected extended (subdivided) --- set 72-73
unmodified immediate --- (set) 57
unreduced --- 28
good count of --- (γ) 28
multiplier of --- (κ) 28
multiplier error of --- (ε) 28
rectangle of --- (r) 28
total count of --- (τ) 28
usefulness-revised --- set 62
--- of iteration I 62
region handling step (second step) of iteration 38
region-usefulness (of a state) 32
region valuation function (ν) 32
regression error (err ρ) 35-36
regression step (of iteration) (third step) 38
regression valuation function (ρ) 33-35
ρ (regression valuation function) 33-35

second step (of an iteration) (region handling) 38
SHRINK (shrinking algorithm) 49-50
similar (regions) 43
state graph 13
final --- (G) 13
final --- set (G) 14
step (of iteration) 14
first --- (solving step) 14
first (solving) --- of iteration I 15
second (region handling) --- of iteration I 38
third (regression) --- of iteration I 38
SUBDIV (subdividing algorithm) 65
subdivided region set 66
--- of iteration I 69

t (total count function) 19
τ (total count of unreduced region) 29
θ (average valuation function) 36-37
total count 19
total error (of an unreduced region) (err) 29
training problem set (P) 14

u (elementary usefulness) 20
uncorrected extended subdivided region set 72-73
unmodified immediate region (set) 57
up (upper point) 17
upper point (of a rectangle) (up) 17
useful 46
usefulness
  average --- (of region set) (avgval) 61
  average --- (of a state) 36
  elementary --- (u) 20
  predicted --- (of a state) 35
  --- of a reduced region (val) 30
  --- of an unreduced region (val) 29
  total --- error of a reduced region (err) 30
  total --- error of an unreduced region (err) 29
usefulness-revised region set 62

val (usefulness of a reduced region) 30
val (usefulness of an unreduced region) 29
valuation function 31-37
  (average) --- (θ) 36-37
  region --- (ν) 32
  regression --- (ρ) 33-35
  --- of iteration I 38

BIBLIOGRAPHY

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

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

Hartigan, J.A.: Clustering Algorithms, Wiley, 1975.

Michie, D. & Ross, R.: "Experiments with the Adaptive Graph Traverser", Machine Intelligence 5, American Elsevier, pp. 301-318, 1970.

Nilsson, N.J.: Problem Solving Methods in Artificial Intelligence, McGraw-Hill, 1971.

Rendell, L.A.: 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, Dec. 1977.

Slagle, J.R. & Farrel, C.: "Experiments in Automatic Learning for a Multipurpose Heuristic Program", Comm. ACM 14, 12, pp. 91-99, Feb. 1971.