A LOCALLY OPTIMAL SOLUTION OF THE FIFTEEN PUZZLE PRODUCED BY AN AUTOMATIC EVALUATION FUNCTION GENERATOR

by
Larry Rendell

Research Report CS-77-36

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

December 1977

ABSTRACT

An outline is given of a system which automatically generates evaluation functions for state-space problems. Initially, the system must be presented with a set of features (functions over states) as input. After a graph traverser attempts a set of problem instances, the system procedes to cluster probability estimates in the feature space, via an effective splitting algorithm. From the clusters, parameters for a (not necessarily) linear evaluation function are computed. In post-initial iterations, the system's graph traverser utilizes an evaluation function generated by the preceding iteration, and, in these later stages, the system refines established clusters both by revising previous probability estimates, and also by further splitting, to effect successively better evaluation functions.

The clustering algorithm is approximately linear with respect to the number of features, so larger sets of features can be handled.

At present the system incorporates a one-way graph traverser, and typically, with useful feature sets, creates evaluation functions which solve every instance of a random sample of fifty fifteen puzzles with an average of fewer than four hundred nodes developed. In experiments to date, the system generally has selected the more useful features from the input set. Futhermore, in the cases which have been examined in detail, the system has chosen values for the parameters which are locally optimal or very nearly so.

INTRODUCTION

This system automatically generates evaluation functions for state-space problems whose instances can be selected or generated according to (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 space can be extended at the beginning of any complete iteration, but in this paper we shall simplify and deal only with feature sets which have been fixed for an entire series of iterations.

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.

SØLVING 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. (The specific form of this evaluation function will be described in a later section). An attempt is halted if a preselected maximum number of states is generated; but whether or not a solution is found, a final description tree 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 description tree 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 description trees. 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 description trees 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). Notice that the purpose of the trees is to allow discovery of these solution path states. In the practical implementation, the trees are released right after a solution is traced and the counts have been determined. Only the feature space points and their associated counts are required from here on.

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 <u>elementary usefulness</u> (<u>of a point, for a particular problem instance set and evaluation function</u>). The assumption of general applicability is presumptuous, since we shall in fact be generalizing from simpler to harder problem instances; however, as the reader will see later on, constant feedback and revision tend to correct biases.

In the section which follows, we shall elaborate on this elementary usefulness, considerably, as we consider how to cluster points and their counts according to their feature space proximity, similarity of usefulness, and a reliability estimate. Furthermore, we shall develop methods of revising clusters, usefulness, and error estimates, as part of the entire "bootstrap" operation which uses the evaluation function to increase efficiency of solution, and the solution results to improve the heuristic.

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 not be detailed here, including a systematic bias in later iterations. For our present purposes it is enough to know that we can estimate a combined (<u>usefulness</u>) <u>error</u> (<u>for a cluster</u>). Generally, the combined cumulative 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 [see Hartigan]. 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:

$$\text{distance}(r_1, r_2, P) \underset{\text{def}}{=\!=} \frac{u_1/(1+e_1)}{u_2.(1+e_2)}$$

where P is the set of points with their counts, $r_1 \cup r_2$ is the rectangle enclosing P, u_i is the usefulness of r_i, e_i is the error factor for u_i $(i=1,2)$, and $u_1 \geqslant u_2$.

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, then 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. (Obviously the algorithm must halt).

The output from this clustering algorithm is a set of clusters whose rectangles constitute a partition of the input rectangle, and whose count functions define both an elementary usefulness and an error estimate for their rectangle.

Notice that the count functions 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 count functions (data), not by the algorithm alone.

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

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 count functions. 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, and whose usefulness and error are undefined. For all later iterations, the input region set is that from the preceding 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 input 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 count functions are $(g_1, t_1), (g_2, t_2), \ldots, (g_m, t_m)$. Then each new region R_i $(i \leqslant m)$ spawned from R will have a usefulness $(g_i/t_i).k$ where $k = u/(\Sigma g_i/\Sigma t_i)$. (Often $k \ll 1$, as we shall discover soon). The new error values are also derived from e and these count functions; the details will not be given here; however it can be seen that the errors will be greater than e.

* There is also a set of feature space parameters which is input, generated from the yet to be discussed third, or regression step. This parameter set plays a minor role, and will not be further mentioned here.

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 the 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 feature area usefulness.

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). Now, especially if $u_1, u_2 \ll 1$, 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

* Actually, this takes place before any splitting occurs.

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 not be explained here, but for our purposes it is enough to know that a revision does take place, and that the new value of usefulness becomes a sum (weighted according to errors) of the old value and a corrected current elementary usefulness value. Usefulness is a product of experience over all the iterations. This revision has an effect of decreasing error estimates and, of course, allowing some readjustment of earlier usefulness estimates.

REGRESSION STEP

After the number of regions increases beyond the dimension, n, of the feature space, 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 n+1 tuples 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 [see Draper & Smith] 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.

EVALUATION FUNCTION

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 rectangle 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.

The above description of the system omits some details. Particularly for the region handling step, there are a number of subtleties. For example, after splitting occurs, the rectangles are allowed to shrink to enclose the feature space points just minimally. Also, there is a bias correction for the elementary usefulness whenever the rergession model is part of the evaluation function, and there are a number of contributions to the usefulness error. Another fact not previously mentioned is that the clustering algorithm can restrict the allowable number of splits in any one dimension to a specified maximum (spacing evenly). But the major aspects have been outlined.

SYSTEM EFFICIENCY and COMBINATORIAL PROPERTIES

A logical question at this point is: Is the system efficient enough to be very useful? The answer is definitely yes. In the solving step, keeping track of feature space values and then mapping states to feature space points requires a low overhead, which increases no worse than linearly with the number of features. Also the number of points is manageable; there can be no more than the total number of developed nodes, and a couple of thousand present no difficulty.

The region handling step is fast, compared with the solving step. It requires less time unless the number of features and possible values for each are quite large. For a typical case with ten features, the region handling step takes roughly as much time as the solving step.*

Because regions are rectangular and aligned with the feature space axes, the clustering algorithm has to deal with a very limited number of tentative splits. Furthermore, the number of partitions increases just linearly with the sum, over each feature, of the number of values each feature can take. Thus the time increases only linearly with the feature space dimensionality (although increasing the dimensionality also tends to increase the actual number of points present, so the net effect can be somewhat worse than linear).

As the number of regions increases with repeated iterations, there are two or three factors affecting the change in speed of execution of the region handling step. On one hand, the time might be expected to increase, since the combined total number of possible partitions increases (but possibly not as fast as linearly, since the size of an average region tends to become

* These figures are based on experiments with the fifteen puzzle.

smaller as their number increases -- although the harder problem instances presented for solving later on tend to expand the space). On the other hand, however, the number of points enclosed by any one region tends to decrease, so this factor tends to improve the speed. Again, though, the total number of points often increases, since the space tends to expand. All in all, the speed seems to decrease, but just slowly (sometimes there has even been an increase), as the regions proliferate. For cases in which a large number of features, some with many values, are used, the clustering algorithm can space its tentative partitions more sparsely. This generally has little or no detrimental effect, and increases the speed markedly.

The regression step requires a completely insignificant amount of time.

TWO EXAMPLES with the FIFTEEN PUZZLE

The first chalenging state-space problem that has been tackled is the fifteen puzzle. The two example iteration series which will be described used similar feature sets; the six features involved were:

f_1 - distance score	(sum of distances of each tile from "home")
f_2 - gnomon blocked score	(Consider the left column and upper row as a single sequence, with the upper left tile in the middle, and count one for each occurrence of one or two alien tiles intervening between correctly placed tiles.)

f_3 - 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_4 - line wrong score	(Examine each line and count one if all the tiles of the line are in in their line, but not in the correct order.)
f_5 - 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.)
f_6 - reversed score	(Count one for each occurrence of two tiles being correctly placed in a line except for a reversal.)

f_1, the distance score, is a feature which has often been used for the fifteen puzzle [Nilsson] and f_6 was used in [Doran & Michie]. f_2 was inspired by the results of a program designed for the fifteen puzzle which first placed the upper left gnomon tiles correctly [Chandra]. The other features were chosen to measure some "things that can go wrong" attribute, like f_6. f_3 is a generalization of f_6 and of f_4. f_2 is essentially a special case of f_5.

The three tables below summarize the results for example one, a series of seven iterations which uses all six of these features. Table 1 lists information about the particular sets of puzzles which were input in each solving step. The "maximum depth" column gives an upper limit for the depth, or minimal path length to the goal. The exact minimal path length is not known, but a maximum value is, because these problem instances were generated randomly to a preselected depth (and later recategorized if a shorter solution arose). At the outset, the hardest puzzles which can be solved have a depth of about nine. Except for the first iteration, a maximum of 1500 nodes was allowed. In the "average number developed" column, only a lower limit is sometimes known, since some of the puzzles are not solved before the systen gives up; and accordingly, the "average path length" is approximate where indicated. From a couple of hundred to a thousand feature space points resulted from the solving step (increasing in later iterations).

TABLE 1 SERIES ONE SOLVING STEP

Puzzles in Input Set

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

Table 2 summarizes the results of the region handling step. Most of the splits which occurred were "expected" and a few were "unexpected", or contrary to the general linear trend. These two categories are listed separately. Below each "feature number", the "expected" splits are to the left, and the "unexpected" to the right. A blank indicates zero.

TABLE 2 SERIES ONE REGION HANDLING STEP

Record of Splits for each Iteration

Iter-ation	Feature Number												
	1		2		3		4		5		6		total
1	2												2
2	3		2		1								6
3	2			1			2				1	1	7
4	1				1						2	1	5
5			3	2	1				1			1	8
6	1						1		5		1		8
7	1			1	2	2					5		11
total	10		5	4	5	2	3		6		9	3	48

Rather than a linear model, a log-linear one was selected for the regressions, so that (neglecting any direct contribution from the regions) the usefulness estimate is $\exp(\rho_0 + \Sigma \rho_i f_i)$, where n is the feature space cardinality, and ρ_i is the parameter for feature f_i. Table 3 lists the parameters which the regression step calculated. Blanks denote zeroes, and the entries are bracketed where the number of regions has not yet reached the number of features (so the parameters are in fact not used). The "error" column lists the regression errors. For each iteration, a confidence level of 0.85 was used.

TABLE 3 SERIES ONE REGRESSION STEP

Parameters Computed

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

Notice that the non-zero parameters calculated by the system do not necessarily conform strictly to the number and dimension of the splits of the corresponding region handling step. In fact, a parameter can be non-zero despite no split having occurred in the corresponding dimension (e.g. iteration 2). This is because some of the features are interrelated, and the shrinking that takes place after the splits reflects this fact.

A second example required fewer iterations before an equilibrium was reached. It used just four of the above features and an entirely different set of input puzzles. The results are summarized in tables 4, 5 and 6 below, which are analogous to tables 1, 2 and 3.

TABLE 4 SERIES TWO SOLVING STEP

Puzzles in Input Set

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

TABLE 5 SERIES TWO REGION HANDLING STEP

Record of Splits for each Iteration

Iter-ation	Feature Number						
	1	3	4		5		total
1	2						2
2	2	1					3
3	1		1	1	3	1	7
4	1	2					3
5						1	1
6							0
total	6	3	1	1	3	2	16

TABLE 6 SERIES TWO REGRESSION STEP

Parameters Computed

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

We shall shortly look into the effectiveness of the evaluation functions generated in these iteration series, but first let us examine some of the properties exhibited by the system, at least in the context of these two examples. These two series of iterations (and others) show the general stability of the system. Both the usefulness and parameter values tend to change just gradually from one iteration to the next. Tables 2 and 5 seem to indicate that the region handling step does succeed in revising usefulness figures toward their "correct" values, since the regression error decreases whenever splitting does not predominate. This correction amounts to a stabilizing negative feedback.

There is also a consistancy. Series two gives results which are very similar to those of the first example. (Compare tables 3 and 6).

There are some other tendencies: Often, when a parameter is zero for an "important" feature, an iteration causes splits in that dimension, and the parameter becomes non-zero (e.g. series one, iteration 6). This is what one would expect. The reverse also applies: Unimportant or redundant features, for example f_6, are rejected. (For feature sets without the more general f_3 and f_4, however, the system accepts f_6). (In passing, it can be mentioned that there seems to be a correlation between how "important" a feature is and both the number of splits which arise and the sparsity of conflicting splits; this will be evident after examining tables 2 and 5 in the light of the graphs which appear later). As also would be expected, it seems that more iterations are required when the number of features is higher.

The following table summarizes results of using some of the evaluation functions which were generated in both of the two series (polynomial part only, omitting the region component). The figures show the performance of the functions with a random sample of 32 puzzles (except series one iteration seven, whose function was given fifty puzzles), allowing a maximum of 2200 nodes to be created (i.e. about 1100 developed). As in tables 1 and 4, any unsolved puzzles count as having >1100 nodes developed, so the average values are sometimes approximate.

TABLE 7 PERFORMANCE of FUNCTIONS

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

From tables 1, 4, and 7, it can be seen that the performance improves very quickly at the beginning, then more slowly. The solving step of series one, iteration 2 had as an evaluation function only the three discrete categories (regions) from iteration 1; no regression polynomial participated; yet half of the puzzles with depth fifteen were solved. In series two, the polynomial provided by just iteration 2 solved three quarters of the completely ramdom puzzles, while the evaluation function of series one, iteration 2 solved nearly all of the standard set. However, excellent performance was not attained until iteration 7 (series one).

To the knowledge of the writer, no other one way graph traverser has solved the fifteen puzzle. An early report [Doran & Michie], in which the experimenters themselves varied the parameters, showed the applicability of features; with the best choice, the program succeeded in solving sixty percent of the puzzles. A two way traverser has solved the fifteen puzzle [Chandra], but with about the same number of developed nodes. Furthermore, that program was designed specifically for the fifteen puzzle, whereas this system was created to work generally, with any state-space problem whose instances can be gotten according to approximate difficulty, for which a meaningful set of features can be supplied, and for which a goal state can be explicitly defined.

In addition, the system has calculated parameters which are optimal (series one) or else very close to optimal (series two).

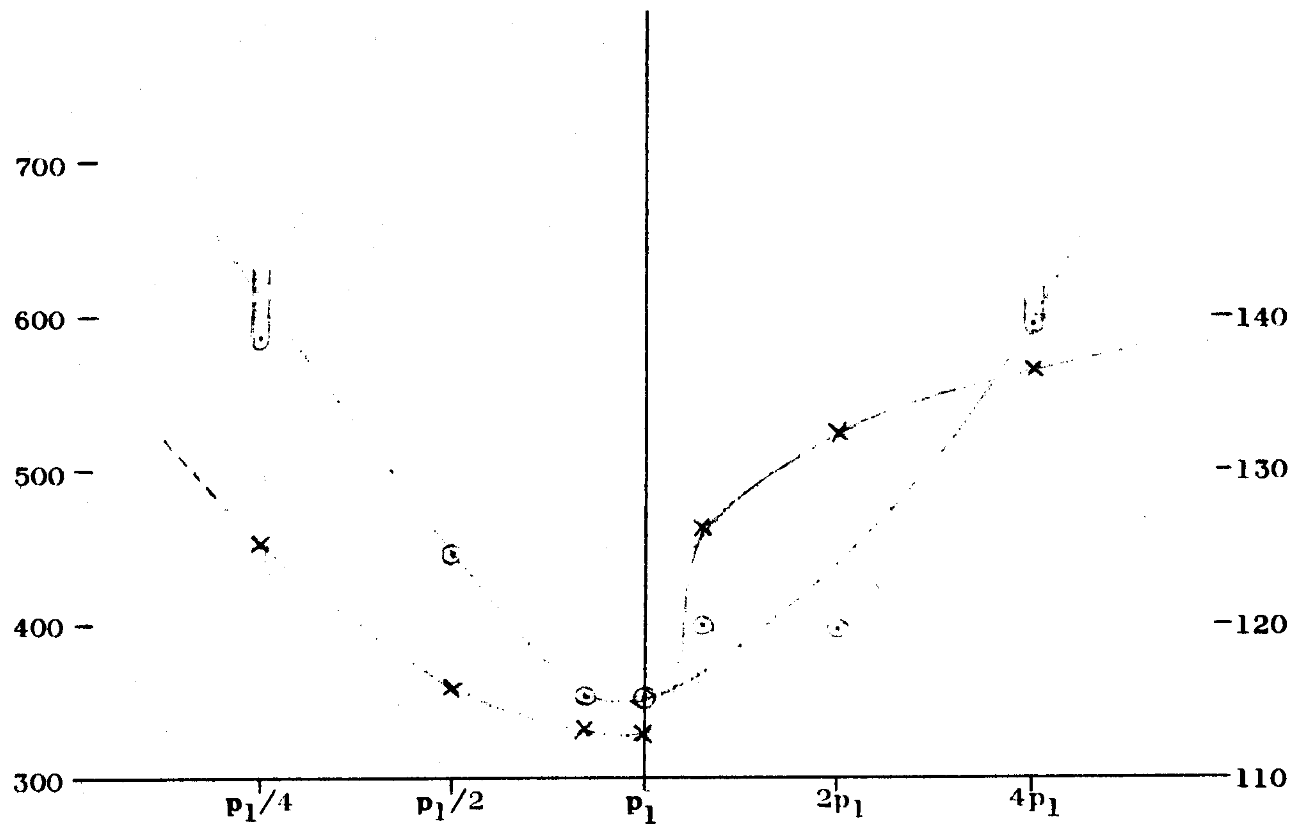
To test this, the parameters from iteration 7 of example one were varied one at a time (actually those for f_3 and f_4 were varied together; f_3 and f_4 are similar) and the resulting linear polynomial was used as an evaluation function with the standard set of thirty-two random puzzles (again with a cutoff after 2200 states created -- or about half that number developed). The average path length and number of developed nodes are graphed below. Both the path length and number of nodes devoloped are plotted on each graph; the numbers to the left refer to average number of developed nodes, and those to the right, to average path length. Dots with circles represent nodes developed. (Open circles indicate the lower limits in cases where not all puzzles were solvable). And x's represent path lengths. (Typical estimates of the standard deviations were one to two hundred for developed nodes and twenty-five to thirty for path lengths).

Although series two did not produce parameters which are identical to those of series one, the only discrepancy arises on the flat portion of the f_3, f_4 curve, and this is only a little above the minimum.

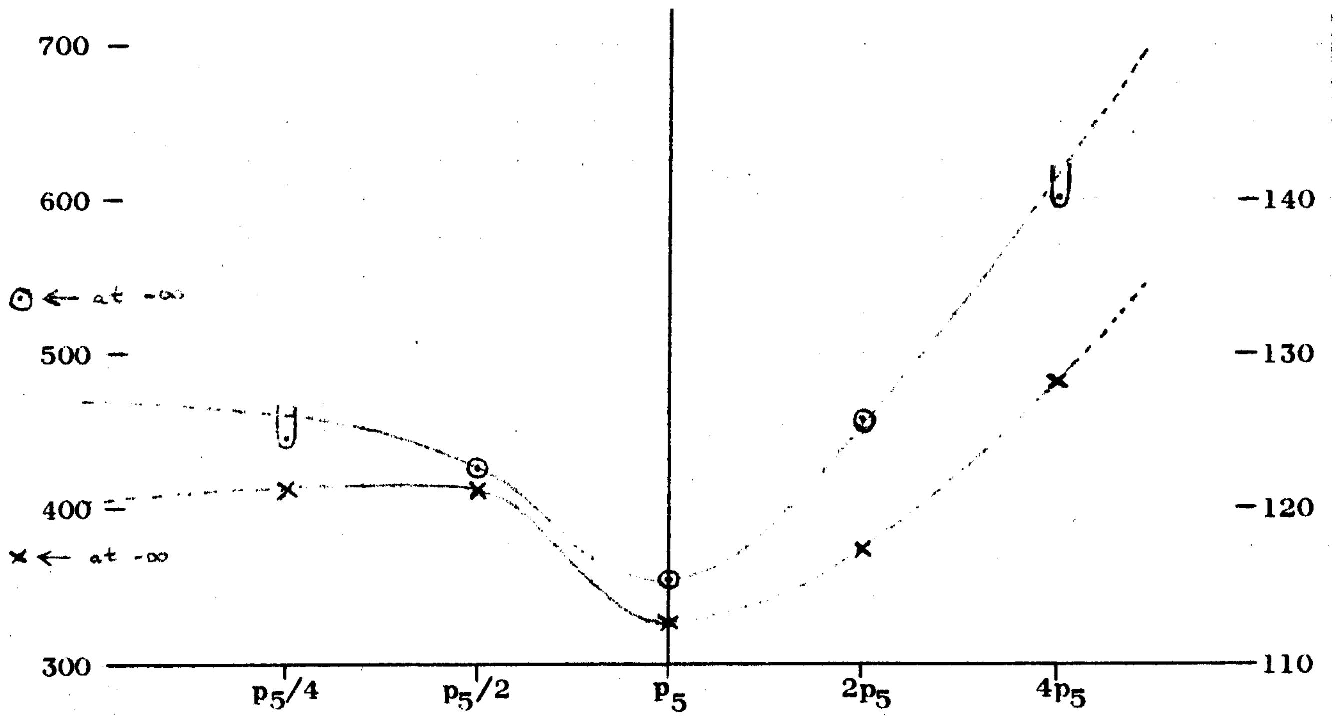
The figures are not given here, but a few program runs with non-zero parameters for f_2 and f_6 were tried, and no improvement could be found. In general, there seems to be very good agreement between the features which the system selects, the parameters it chooses, and the actual performance of the resulting evaluation functions.

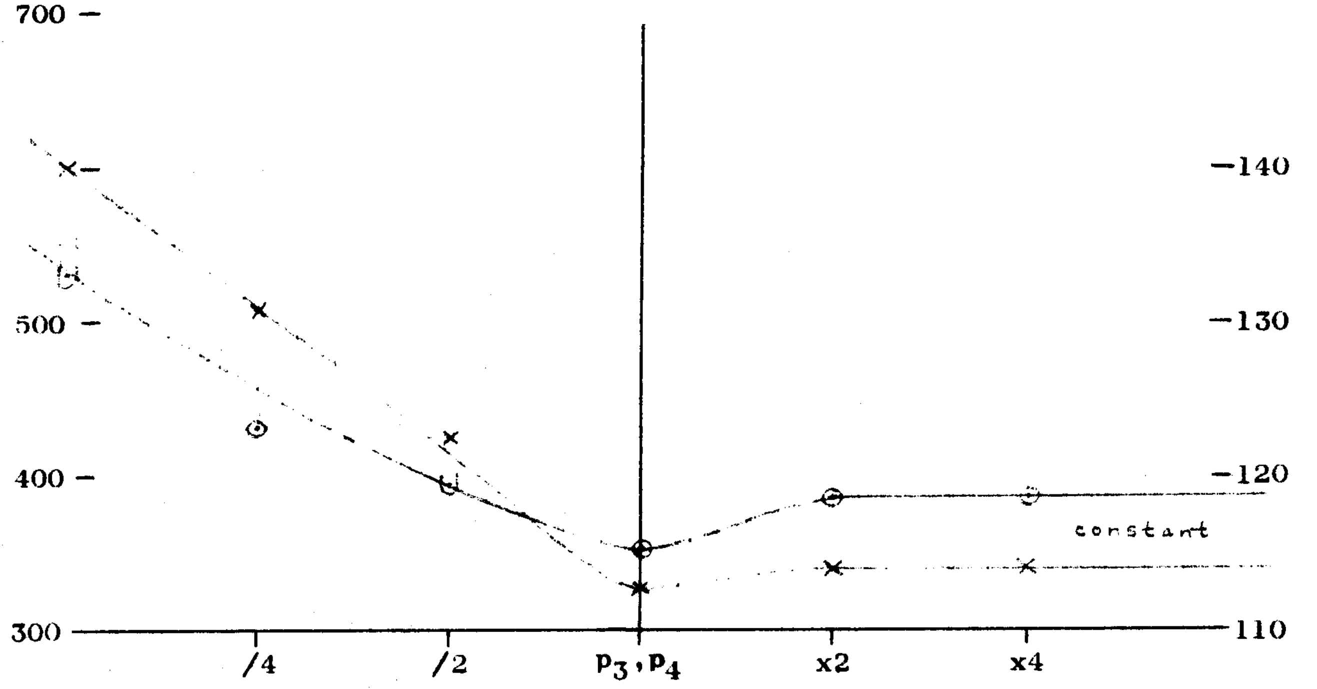
Performance of System-Generated Evaluation Function
(Series One Iteration 7) with Disturbed Parameter Values
(Using Standard Set of Thirty-Two Puzzles)

Average Nodes Developed (circled points) and
Average Path Lengths (crosses) vs Parameter 1
(Center Line Represents Value Computed by System)



Average Nodes Developed and Path Lengths vs
Parameter 5 (upper graph) and Parameters 3 and 4 (lower graph)





BIBLIOGRAPHY

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

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

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

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

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