PREDICTING THE LENGTH OF SOLUTIONS TO PROBLEMS

J.R. Quinlan
Basser Department of Computer Science
University of Sydney, Australia

Introduction

This papier is concerned with a class of problems whose solutions are in a sense linear. Several formalisms have been used to describe such problems [1,2,3,4,7,9], but the central ideal can be simplified and summarised as follows The universe is a network consisting of a set $S=\{s_i\}$ of *objects* connected by a set E of *edges*. A *problem* is the task of finding a path from one object s_i to another object s_j. A solution consists then of a sequence of objects

$$s_i = s^*_0,\ s^*_1,\ \ldots,\ s^*_{n-1}, s^*_n = s_j$$

such that an edge connects each object to the next. The above will be called a *solution of length n* because the path from s_i to s_j contains n edges.

Several systems have been developed for discovering solutions to these problems. They have one feature in common with most algorithms in artificial intelligence, namely: at any stage in the search for a solution, a choice must be made among several possible next actions. This choice is known to be crucial, for if it is made felicitously a solution may be found quickly, whereas a great deal of time may otherwise be spent fruitlessly. Many successful systems incorporate relatively complex mechanisms to make this choice [5,10,14] but these mechanisms often seem to be bound inexorably each to its own system, and thus cannot be transferred to new systems.

This situation was relieved by analysis [7,8] indicating that the ability to make a good choice comes down to (in our simple formalism) the ability to predict the lengths of solutions to problems before they are solved. This is a particularly useful approach because the length of a solution to a problem does not depend on the way that the solution was found, and so the choice-making function can be divorced from the algorithm proper. In particular it is possible to study the prediction task in isolation, so long as the implications of the eventual reunion of all the components are borne in mind.

The following section develops a general prediction scheme that could be adapted to any system for handling this type of problem. This is followed by an examination of its performance on a substantial number of problems.

A General Scheme

The underlying structure of the scheme is straightforward. Suppose we have a set of problems whose solution length is known, and an unsolved problem whose solution length we wish to estimate. The approach taken is to predict that the unsolved problem will have the same solution length as the 'most similar' solved problem. To determine the most similar solved problem, we will map all the problems into points in some space and find the point closest to the unsolved problem. The main part of the scheme is concerned with setting up the mapping in such a way that points in the space which are close together do tend to have similar solution lengths. Note that the converse is not true: we do not require that all points with similar solution lengths lie close together. For instance, the points with a given solution length may form several distinct clusters throughout the space.

A few preliminary remarks must be made. A problem may have many solutions of various lengths, so 'the' length will be taken to be the shortest. Alternatively a problem may have no solution, in which case we take the length to be a large number. Finally, while a solution length is clearly an integer, we will allow an estimate to be a real number such as 3.7, which may be interpreted as 'closer to 4 than to 3'.

The first step will be to look at the differences between a pair of objects. We are after some means of describing the changes that must be wrought on the first to make it identical to the second. Clearly this description will depend on the nature of the objects and cannot be defined in general terms. We thus postulate the existence of a <u>difference function</u>

$$\delta : S \times S \to R^q$$

which, given a pair of objects, produces a vector of q non-negative real numbers. Section 3 describes two such functions that may serve as illustrations.

Now *6* maps a pair of objects into R^q, and a problem consists of finding a path between a pair of objects s, and s., say. It might seem that δ alone will suffice to map problems into a suitable space. However, recall that we will need to find the closest point to an unsolved problem, so we will need to use distance in the space. The various components of $\delta(s_i,s_j)$ might measure quite different things, with the result that using δ alone might render distances in the space meaningless. Consider the analogy of mapping people into R^2 by taking their height in feet and their weight in pounds. Suppose we have two individuals who map to <3.5,150.0> and <6.0,160.0>. A third individual <5.9,154.0> would then be closer to the former than the latter! To ensure that distances are meaningful, then, our mapping must have the property that the coordinates of a point are measured in the same units.

Since the length of a problem's solution will also clearly depend on the edges E connecting objects, it would seem desirable to bring them into the mapping somehow. Now each edge e connects two objects s and s , say. We can form
x y
a picture $\delta(s_x,s_y)$ of the difference between the objects connected by edge e. If we could form such a picture for each edge and average them, the result would be a vector

$$\underset{\sim}{d} = (d_1,d_2, \ldots, d_q)$$

where d_i is the average value of the ith component of the difference between pairs of objects connected by an edge. Intuitively, d is a picture of the difference created by moving along a 'typical* edge. Each component of δ is non-negative, so d. will be non-negative. Moreover, d. will be zero only if the ith component of the difference between all pairs of objects connected by edges is zero (in which event the ith component of the difference function could well be scrapped). But for most problems of interest S and E are very large or even infinite, so d cannot be computed as above. Here we make use of the fact that the scheme is to be embedded in a problem-solving system. It is most unlikely that a problem-solver would be set up to tackle a single problem in some universe; even if it were, it would solve some subproblems in the course of its search for a solution. We can assume, then, that a number of problems with identical or very similar S and E have been solved. The vector d can be approximated by averaging $\delta(s_x,s_y)$ not over all edges e, but over those edges used in solutions to the previous problems.

Again, a problem is solved when a path is found from object s_i to object s_j. If

$$\delta(s_i,s_j) = (x_1,x_2, \ldots, x_q)$$

then x_k. is the kth component of the difference that will arise as we move from s. to s,. Aqain,

d is the corresponding component of the difference that arises in moving along a typical edge. The ratio x_k/d_k. is then the number of typical edges that we would expect to move along in getting from s. to s_i.. If we form

$$\underset{\sim}{z} = (x_1/d_1,x_2/d_2, \ldots, x_q/d_q)$$

then z_k is a crude estimate, derived from the kth component of the difference function, of the length of the solution to the problem. Each component of z is clearly measured in the same units, so mapping problems to their z descriptions in R^q will result in meaningful distances in the space.

At first glance, it seems that we have changed the task of predicting the length of solutions into a classical pattern recognition problem [6] in which the z's are the description vectors and the classes are the solution lengths. This unfortunately is not so. Since the picture of a problem is incomplete, it is quite possible to envisage two problems with the same picture but different solution lengths. Secondly, we have allowed the prediction function to take real values so the number of classes would become infinite. As a matter of peripheral interest, when the first set of data from the next section was set up as a pattern recognition problem using a clustering algorithm [13], the results were poor.

Still, the approach taken is a variant of a familiar pattern recognition technique. Each z is clearly a point in q-space, so we imagine a set {p.} of prototype points embedded in this space. Each prototype point p. has associated with it a frequency f_j. and a class sum c , both of which are integers. As usual we operate in two modes, classifying and training. For classification we are given to description z of a problem; we find the prototype point p. nearest to z and predict the solution length as the ratio c./f.. in training we have a description of a problem with known solution length n. If there already exists a prototype point p_j that is very close to $\underset{\sim}{z}$, i.e., $|\underset{\sim}{p}_i - \underset{\sim}{z}| \leq$ some ε, we modify it by

(i) moving it slightly: $\underset{\sim}{p}_i$ becomes

$$\frac{f_i \underset{\sim}{p}_i + \underset{\sim}{z}}{f_i + 1}$$

(ii) changing its frequency and class sum; f. becomes f +1 and c. becomes c.+n.

If no prototype point is very close to z, we enter the latter as a new prototype point $\underset{\sim}{p}_j = \underset{\sim}{z}$, setting f . to 1 and c. to n.

One matter remains to be dealt with. If the predictor is to be useful in practice it cannot be allowed to acquire prototype points without limit. There must be some mechanism for reducing their number so that the performance of the predictor is degraded as little as possible. This implies that some means of generalizing a subset of the prototype points is required, in the vein of Samuel's generalization of checker positions [11]. A simple process is defined that reduces by one the number of prototype points. Basically, it looks for two neighbouring points with similar class sum/frequency ratios and replaces them by an average point.

We say two prototype points p. and p. are adjacent if the distance between them is less than the distance from either to any other prototype point. We consider merging them to form a single prototype point $\underset{\sim}{p}_k$, with

$$f_k = f_i + f_j$$
$$c_k = c_i + c_j$$
$$\underset{\sim}{p}_k = \frac{f_i \underset{\sim}{p}_i + f_j \underset{\sim}{p}_j}{f_i + f_j}$$

Now this merge has associated with it a cost in the form of a change in class sum/frequency ratios that can be expressed as

$$f_i \left| \frac{c_k}{f_k} - \frac{c_i}{f_i} \right| + f_j \left| \frac{c_k}{f_k} - \frac{c_j}{f_j} \right|$$

which reduces to

$$2 \frac{|f_j c_i - f_i c_j|}{f_i + f_j}.$$

The procedure then consists of examining each pair of adjacent points and merging that pair for which the above cost is minimal. Each application of the procedure reduces the number of prototype points by one. Section 3 reports on the degradation of prediction performance as the number of prototype points is reduced.

Early in this section we postulated a function 6 for producing a vector description of the changes that would have to be made to one object in order to make it identical to a second. This function eventually yielded a description z of a problem in q-space. The scheme proceeded under the assumption that two problems with points lying near each other will tend to have solutions of similar length. In other words, we required 6 to provide a meaningful description so that this spatial property comes about. Now for some problem areas with built-in spatial connotations (such as transportation problems) it is not difficult to formulate such a 6. But what about the rest? The next section examines two universes, one finite and one infinite. In each case it is possible to develop a simple 6 so that the prediction scheme gives useful results. Hopefully these successes indicate a general applicability of the scheme.

Testing the Scheme

In the first universe objects are binary trees, a representation recognized as being both common and general [7]. The universe is infinite, so objects and edges cannot be stated explicitly. Edge information is given by <u>rewriting rules</u>, each of the form $s_\ell := s_r$ where s_ℓ and s_r are objects. The meaning of such a rule is as follows: if s is an object containing an instance of s_ℓ then s. can be rewritten as the object s. formed by replacing the instance of s_ℓ with a corresponding instance of s . For example, consider the object (a+b)+c and the rule x+y := y+x. The object is an instance of x+y, and so can be rewritten as the corresponding instance of y+x, namely c+(a+b). But a+b is also an instance of x+y, so the original object could also be rewritten by this rule as (b+a)+c. An edge connects s. to s_j if and only if s. can be rewritten as s,.

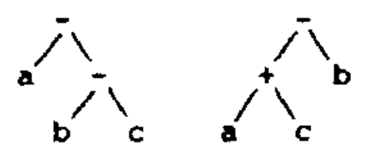
Since S is infinite, each rewriting rule normally defines an infinite number of edges.

Two sets of problems were established. The first (set A) contained 152 problems relating to the manipulation of algebraic expressions involving + and -. The number of rewriting rules varied from 6 to 23, solution lengths from 1 to 6. The second (set S) of 139 problems was concerned with establishing the equivalence of programs using a formal language to describe flowcharts [12]. The number of rules ranged from 26 to 40, solution lengths from 1 to 5. Sample problems and solutions for sets A and S appear in [9,10]. A number of problems from both sets have been presented to human problem-solvers who found them non-trivial.

A moderate amount of experimentation was needed to find a suitable 6 (largely because the early attempts were too sophisticated). It was decided to use only syntactic information, i.e., to disregard entirely the meanings of objects represented as trees. Further it was decided to ignore the difference between operators and operands. Six operations for manipulating binary trees were formulated, as

$\nearrow$:	move a symbol up and to the right
$\nwarrow$:	move a symbol up and to the left
$\searrow$:	move a symbol down and to the right
$\swarrow$:	move a symbol down and to the left
$\oplus$:	add a new symbol somewhere in the tree
$\ominus$:	delete a symbol somewhere in tree.

Any changes to a binary tree can be described using these operations. The function 6 produces a vector of 6 elements, one for each of the operations. The kth element of $\delta(s_i,s_j)$ is simply the number of times that the kth operation above was used in changing s_j to s_j . The uniqueness of the vector was ensured by a number of arbitrary restrictions, such as only using the operations $\oplus$ and $\ominus$ when the move operations by themselves were inadequate. For example, consider the two objects



which we denote by s_1 and s_2 respectively. To change s_1 to s_2 we must

(i) move a down and to the left ($\swarrow$)

(ii) move c $\nwarrow$ twice, and then $\swarrow$ and $\searrow$

(iii) move b $\nearrow$

(iv) add a symbol + to s_1

(v) delete a symbol - to s_1.

The vector produced by counting the number of times each operation was used is

$$\delta(s_1,s_2):\ \begin{array}{cccccc} \nearrow & \nwarrow & \searrow & \swarrow & \oplus & \ominus \\ \hline 1 & 2 & 1 & 2 & 1 & 1 \end{array}$$

This δ is completely independent of the rewriting rules available to solve a problem; there is no idea of $\delta(s_i,s_j)$ providing a model of how to get from s to s_j. The same δ was thus used for both sets of problems. The 'typical' edge difference d for each problem was computed from the solutions to previous problems in its set.

The second universe was that of the 8-puzzle. An object consists of a 3x3 arrangement of tiles bearing the digits 1 to 8 and a blank tile. An edge exists between two objects if the second arises from the first by interchanging the blank tile with one of its neighbours. In this problem domain both S and E are finite, though large (S contains about 360,000 objects).

Three measures were used to set up δ, all of them adapted from those mentioned by Nilsson [7], We define $\delta(s_i,s_j)$ as (x_1,x_2,x_3) where

x_1 = the number of non-blank tiles that occupy different positions in s_i and s_j

x_2 = the sum of the cell distances apart of the above different positions

x_3 = going around the edge, the number of non-blank tiles that have different successors in s_i and s_j.

For example, if s_i and s_j are the objects

```
1  5  2      1  2  3
4  8         4     5
6  3  7      6  7  8
```

then $\delta(s_i,s_j)$ is (5,9,5). (Note that, unlike the first δ, this one has a substantial connection to the way a problem is solved; in fact, x_1 is a lower limit on the length of the solution.) The third set (set P) contained 186 8-puzzle problems with solution lengths ranging from 1 to 18.

The basic measure of the predictor's performance on a problem was taken to be the <u>absolute error</u>, i.e., the magnitude of the difference between the actual and predicted solution lengths. (This follows the approach to errors in heuristic functions in [8].) For a set of problems the performance measure was the average absolute error on the set. The predictor could be considered useful in so far as it outstrips the obvious random procedure of selecting an arbitrary length from those actually occurring in the set. This random procedure has an expected average absolute error of

2,02 on set A (selecting a length from 1 to 6)
1.63 on set S (ditto from 1 to 5)
6.10 on set P (ditto from 1 to 18).

The first experiment was designed to test the scheme in an environment similar to that which would be encountered in practice. Each set was first shuffled so that the original order of the problems would have no bearing on the outcome. Each problem in a set was first tested, the absolute error found, and then the problem used to train the system. Each problem was thus tested with the predictor having the benefit only of experience from previous problems in the set. Three average absolute errors were computed: for the first 2/3 of the problems, for the last 1/3, and for the whole set. This experiment was performed with 20 different shuffles of each set.

The results obtained appear in Table 1. This shows that the performance on the whole set was better than the random procedure in all cases, but particularly so for sets A and P. There is also a clear improvement on the last 1/3 in all sets. Considering the numbers of problems and shuffles, this improvement can only be ascribed to the experience obtained on previous (different) problems.

<u>Table I</u>

Alternating test and train: average absolute errors on 20 shuffles and mean.
The values shown for each set are for the first two-thirds of the problems, the last third, and the whole set.

Set A			Set S			Set P		
1.05	.67	.93	1.08	.92	1.02	1.03	.36	.81
.91	.70	.84	1.13	.85	1.03	.80	.60	.73
.90	.70	.83	1.08	.86	1.00	.73	.66	.70
.85	.87	.85	1.03	.85	.97	.77	.55	.70
.96	.69	.87	1.11	.86	1.03	.87	.56	.76
1.15	.78	1.02	1.02	.75	.92	.80	.61	.74
1.06	.75	.95	1.22	.79	1.07	.80	.43	.68
1.01	.66	.89	1.04	.86	.98	.77	.60	.71
.99	.80	.93	1.22	.98	1.14	.78	.85	.80
.91	.84	.89	1.05	.81	.97	.68	.57	.64
1.18	.56	.97	.95	.84	.91	.82	.67	.77
.91	.85	.89	1.07	.81	.98	.73	.48	.65
.84	.85	.84	1.07	.87	1.00	.79	.48	.68
1.07	.75	.97	1.16	.75	1.02	.72	.65	.69
.90	.74	.85	1.02	.96	1.00	.64	.70	.66
.90	.85	.89	1.10	.84	1.01	.77	.53	.69
.83	.85	.84	.97	.95	.96	.68	.62	.66
.92	.71	.85	1.19	.80	1.05	.74	.52	.67
1.00	.72	.91	1.08	.92	1.02	.64	.60	.63
1.00	.73	.91	1.11	.84	1.02	.73	.72	.73
Means:								
.97	.75	.90	1.08	.85	1.01	.76	.59	.71

The second experiment was drawn up to investigate the predictor's performance on the sets of problems after training, and to see how it was affected by reduction of the number of prototype points. The prototype points developed in the first experiment were retained and each problem again presented for classification. The number of prototype points was then reduced by repeated merging and each problem again fed to the predictor. This reduction process was performed several times.

The results appear in Table II. It is clear, first of all, that the predictor is doing very well with its full complement of prototype points. Secondly, it is possible to merge quite a large number of prototype points before the performance of the predictor is seriously degraded; even allowing only 10, the predictor still does passably. This suggests that the scheme should still be useful if a restricted amount of space was available to hold prototype points.

Table II

Degradation of retrial performance as the number of prototype points is reduced.

Set A		Set S		Set P	
no. of points	**average abs. error**	**no. of points**	**average abs. error**	**no. of points**	**average abs. error**
126	.15				
120	.15	117	.20		
110	.18	110	.20		
100	.18	100	.20		
90	.19	90	.20		
80	.23	80	.24		
70	.30	70	.29	64	.32
60	.35	60	.37	60	.32
50	.45	50	.47	50	.32
40	.61	40	.58	40	.32
30	.73	30	.66	30	.37
20	.83	20	.78	20	.45
10	.91	10	.91	10	.63

Conclusion

On the basis of its performance on the sets of problems in the previous section, it seems fair to say that the scheme gives useful results, even with an unsophisticated 6 function. It is being included as part of a problem-solver in the process of development. Still, many improvements are possible.

In Section 2 we went to some lengths to ensure that the various components of the z description of a problem were measured in the same units. As a result, distances were equivalent in all dimensions of the problem description space. Ths does not allow for the possibility that small changes in one component could be more significant than large changes in another. A more flexible definition would allow distances in different dimensions to be weighted differently. Experiments have been conducted along these lines using statistics such as the variance of d. from problem to problem in a set. The result was a small but noticeable improvement.

Certain aspects of the scheme may not be appropriate if the number of training points were to run into the thousands. In particular. it may be desirable to impose some sort of ageing process on prototype points and on the solutions used to give a picture of the 'typical' edge. These measures would be designed to make the system more responsive to its recent history rather than its total history in some problem domain.

Finally, there still remains the nuisance of designing an appropriate . This may not be as annoying as it appears. For the universe of binary trees we used only structural information on objects, so the same 6 was appropriate for two quite different problem domains. A relatively small number of structures is used to represent objects in artificial intelligence, so a few 6's may suffice to handle a large number of problem domains. The ideal system, of course, would specify a loose framework for 6 and allow the function itself to be induced from experience, along the lines of a feature extraction exercise. Intriguing as this sounds, it may turn out to be pinning one's hopes on a <u>deus</u> ex <u>machina</u>.

References

1. Chang, C.L. The unit and the input proof in theorem-proving. <u>J.ACM 17</u>, 4 (October 1970) pp. 698-707.
2. Doran, J.E. An approach to automatic problem-solving. In Collins and Michie (eds.J, <u>Machine Intelligence 1</u>. Oliver & Boyd (1967) pp. 65-87.
3. Ernst, G.W. and Newell, A. <u>GPS: A Case Study in Generality the Problem Solving</u>. Academic Press (1969).
4. Loveland, D.W. A unifying view of some linear Herbrand procedures. <u>J.ACM 19</u>, *2* (April 1972), pp. 366-384.
5. Michie, D. and Ross, R. Experiments with the adaptive Graph Traverser. In Meltzer and Michie (eds.). <u>Machine Intelligence 5</u>. Edinburgh Univ.Press (1970) pp. 301-318.
6. Nilsson, Nils J. <u>Learning Machines</u>- McGraw-Hill (1965).
7. ______. <u>Problem-Solving Methods in Artificial Intelligence</u>. McGraw-Hill (1971).
8. Pohl, I. First results on the effect of error in heuristic search. In <u>Machine</u>

Intelligence 5 (loc. cit.), pp.121-134.

9. Quinlan, J.R. and Hunt, E.B. A formal deductive problem-solving system. J.ACM 15, 4 (October 1968) pp. 625-646.

10. ______. A task-independent experience-gathering scheme for a problem-solver. Proc.Int.Joint Conf.on Artificial Intelligence (Washington 1969) pp. 193-198.

11. Samuel, A. Some studies in machine learning using the game of checkers. In Feigenbaum and Feldman (eds.), Computers and Thought. McGraw-Hill (1963), pp. 71-105.

12. Sanderson, J. Theory of programming languages. Thesis, University of Adelaide, Adelaide, Australia (1966).

13. Sebesteyen, G. Decision Making Procedures in Pattern Recognition. McGraw-Hill (1962).

14. Slagle, J.R. and Farrell, CD. Experiments in automatic learning for a multi-purpose heuristic program. C.ACM 14, 2 (February 1971) pp. 91-98.