Acknowledgements

I am grateful to P. J. Landin and C. Strachey for many illuminating discussions about the theory of programming, and to Professor J. A. Robinson for helping me to clarify the ideas presented here. The ideas are based on the work of Professor J. McCarthy. I would also like to thank IBM Corporation for making it possible for me to attend the conference on the Mathematical Theory of Computation and give the talk from which this paper is derived.

References

BARRON, D. W., and STRACHEY, C. (1966). Programming, *Advances in Programming and Non-numerical Computation* (ed. L. Fox) pp. 49–82.

BROOKER, R. A., and ROHL, J. S. (1967). Simply partitioned data structures: the compiler-compiler re-examined, *Machine Intelligence* 1 (eds. N. L. Collins and D. Michie). Edinburgh: Oliver and Boyd, pp. 229–239.

BURSTALL, R. M. (1968). Semantics of assignment, *Machine Intelligence* 2 (eds. E. Dale and D. Michie). Edinburgh: Oliver and Boyd, pp. 3–20.

COOPER, D. C. (1966). The equivalence of certain computations, *Computer Journal*, Vol. 9, pp. 45–52.

COHN, P. M. (1965). *Universal algebra*, New York and London: Harper and Row.

CURRY, H. B., and FEYS, R. (1958). *Combinatory logic*, Amsterdam: North Holland.

FLOYD, R. W. (1967). Assigning meanings to program, *Mathematical Aspects of Computer Science*. Amer. Math. Soc. Providence, Rhode Island, pp. 19–32.

KAPLAN, D. M. (1967). Correctness of a compiler for Algol-like programs, *Stanford Artificial Intelligence Memo. No.* 48, Department of Computer Science, Stanford University.

LANDIN, P. J. (1964). The mechanical evaluation of expressions, *Computer Journal*, Vol. 6, pp. 308–320.

LANDIN, P. J. (1966). The next 700 programming languages, *Comm. Ass. Comp. Mach.*, Vol. 9, pp. 157–166.

MCCARTHY, J. (1963). A basis for a mathematical theory of computation, *Computer Programming and Formal Systems* (eds. Braffort and Hirschberg), Amsterdam: North Holland, pp. 33–70.

MCCARTHY, J., and PAINTER, J. A. (1967). Correctness of a compiler for arithmetic expressions, *Mathematical Aspects of Computer Science*. Amer. Math. Soc. Providence, Rhode Island, pp. 33–41.

PAINTER, J. A. (1967). Semantic correctness of a compiler for an Algol-like language, *Stanford Artificial Intelligence Memo. No.* 44 (March 1967), Department of Computer Science, Stanford University.

RICHARDS, M. (1967), Basic CPL reference manual, Memo. M-352, Project MAC, M.I.T.

Note added in proof. Some further work on the topic of this paper is reported in:

LANDIN, P. J., and BURSTALL, R. M. (1969). Programs and their proofs: an algebraic approach. To appear in *Machine Intelligence 4* (eds. B. Meltzer and D. Michie). Edinburgh: University Press.

A program for solving word sum puzzles

By R. M. Burstall*

This paper describes a program for solving a class of 'word sum' or 'cryptarithm' puzzles by a heuristic tree searching method. Formally the problem is to solve a set of simultaneous linear inequalities with the variables taking integer values.

(First received October 1967 and in revised form May 1968)

1. Introduction

This paper describes a heuristic program for solving a class of 'word sum' problems sometimes called 'cryptarithms' (Brookes, 1964), which have attracted some attention as a simple example of problems which can be solved by brute enumeration but which can better be tackled with heuristic search reducing strategies. An example (with acknowledgements to English Electric) is

$$\begin{array}{r} K\ D\ F\ 9 \\ K\ D\ N\ 2 \\ K\ D\ F\ 6 \\ K\ D\ P\ 1\ 0 \\ \hline C\ A\ R\ E\ E\ R \end{array}$$

Each letter is to be replaced by a different digit to form a correct addition sum.

Enumeration means scanning 10! possibilities. Can the computer adopt a more humane method?

2. Mathematical formulation

We can express the problem thus (adding variables for carries)

$$\begin{aligned} 9 + 2 + 6 &= R + 10v \\ v + 2F + N + 1 &= E + 10w \\ w + 3D + P &= E + 10y \\ y + 3K + D &= R + 10z \\ z + K &= A + 10C \end{aligned}$$

* *Department of Machine Intelligence and Perception, University of Edinburgh*

These equations can be solved by integer linear programming (Gomory, 1963; Beale, 1965), taking any arbitrary expression as the one to be maximised. This is rather a sledgehammer approach and coping with the side condition that variables other than carries must take distinct values involves an extension to quadratic programming or the addition of many extra equations and variables.

3. The search method

The method used here is a tree searching method, alternately making *deductions* based on the known bounds of the variables and *hypotheses* about possible values of the variables.

For example, we see immediately that $R = 7$. The carries must be less than 4, hence $C \leqslant 1$. ($C \neq 0$ by convention.) Continuing thus we obtain $z \leqslant 3$, $K \geqslant 8$, $z \geqslant 2$ and $A = 0$ or 2. Now we are unable to draw further deductions and must make alternative hypotheses e.g. $A = 0$ or $A = 1$. For each case we may continue a separate series of deductions until they dry up and we are forced to resort to further hypotheses and examine sub-cases. At each stage the set of alternative hypotheses must of course cover all the possibilities.

Thus we alternate between a deductive phase, taking a set of bounds for the variables and 'narrowing' them to closer bounds, and a hypothesis-making phase causing us to branch out and develop recursively a tree of sub-problems to be solved.

The method of making hypotheses used was to choose a variable and split its current range in two. E.g. if $0 \leqslant D \leqslant 7$, we could take $0 \leqslant D \leqslant 3$ and $4 \leqslant D \leqslant 7$ as alternatives.

4. The deductive phase

Attempts to narrow the bounds by deduction can result in:

(i) A contradiction, showing that our last hypothesis was untenable. In this case we turn to its alternatives.
(ii) Each variable being restricted to a single value. This gives a solution.
(iii) Inability to make further deductions. Here we are forced to make alternative sub-hypotheses.

The kind of deductions used were simple ones using inequalities. For example we treat the equation

$$3x - 4y + 2z = 11 \qquad (1)$$

as

$$3x - 4y + 2z \leqslant 11 \qquad (2)$$

and

$$3x - 4y + 2z \geqslant 11. \qquad (3)$$

Suppose we know that $0 \leqslant x \leqslant 2$, $0 \leqslant y \leqslant 6$ and $1 \leqslant z \leqslant 5$. Using the upper bounds of x and z with (3) we have $4y \leqslant 5$, i.e. $y \leqslant 1$. In general for a $\geqslant$ inequality we add together the upper bound of each variable (or lower bound if it has a negative coefficient) multiplied by its coefficient and subtract the right-hand side from the sum. This gives us the amount of 'slack' in the inequality. If any variable has a range which when multiplied by the modulus of its coefficient is greater than this slack its range can be reduced. A similar operation is carried out for $\leqslant$ inequalities.

To aid efficiency each inequality is considered in turn and when a deduction takes place only those inequalities which might be affected by the change are put back on the list for consideration.

After each deduction a check is made to see whether the rule that no two letters have the same values can be used.

5. Choosing a variable to split

It may make quite a lot of difference which variable we choose to split in the hypothesis phase. How can we make a sensible choice?

The first method tried, the 'random method', was to try the effect of splitting not just one variable, i.e. one way of dividing up all the possibilities into separate cases, but tentatively splitting several variables one at a time, to see which split is the most profitable in yielding further deductions. For each variable we make all the possible deductions but do not make further sub-hypotheses. Instead we choose the variable which gave most new information on splitting and continue our search by splitting that.

Thus we are not just exploring one tree of alternative hypotheses, but trying out different methods of growing the tree. This is more sophisticated than most tree search programs.

How do we compare the effects of tentatively splitting different variables? An obvious measure was the space-size, i.e. the product of the ranges of the variables, measured on a logarithmic scale. This product is in fact the number of possible solutions remaining. For each variable we add the space-sizes in the two half ranges after narrowing them to get a 'badness'. We choose the least bad variable.

Fig. 1 shows the tree developed in a typical computer run. The vertical lines with branches off them indicate tentative splitting of several variables chosen at random (up to 3 in this case). E, K and another variable were first chosen, but the upper range of K produced a contradiction, so that it was not necessary to try splitting the third variable; a fresh lot of variables could be chosen using only the lower range of K. The next variables chosen for tentative splitting were F, P and N; P was the best of these. From then on we have two sub-problems on our hands to be done one after the other. No further splitting into sub-problems took place after this, because on each tentative splitting one of the variables produced a solution or a contradiction in one of its two branches.

Another slightly more refined method of choosing variables was used, the 'merit method'. In this method some information accumulated during the narrowing phase was used to calculate, for each variable which could be split, and for each of the two branches, the number of variables which could immediately have their bounds reduced in consequence. Adding the number affected for each of the two branches gave a figure of merit for the variable under consideration. This calculation was quite quick, and it allowed an approximate evaluation to be made for each variable instead of a more extensive evaluation for only a few of the variables selected at random. It was now possible either simply to take the most meritorious variable, or to combine the features of both methods by comparing

the 'badness' after narrowing the 2 or 3 most meritorious variables, instead of 2 or 3 chosen at random. This combined method turned out to be the fastest.

6. Computer results

A program was written in ALGOL for the Elliott 503 computer using a recursive procedure to solve each sub-problem. **Table 1** shows five problems which were run (some of them taken from Brooke, 1964). **Table 2** shows the results. Each problem was run five times for each set of conditions using different random numbers. The merit method led to rather shorter computing times than for the random method in spite of the extra calculation involved. Choosing two or three variables for splitting and taking the one with least badness (space-size) was faster than choosing just one, in spite of the extra time spent doing tentative splitting. This showed that this look-ahead feature was worthwhile. The effect was only evident for the random method and not for the merit method, as would be expected since in the merit method the one variable was preselected with a certain amount of cunning.

The program also solved several other word sum problems successfully. The program was extended to maximise a linear function of the variables in order to see whether it could tackle integer linear programming type problems, but the search tree increased too rapidly for it to have much success with even rather easy examples. A more detailed account of the program and results is given in Burstall (1965).

Table 1

Cryptarithm problems

PROBLEM 1

```
  K D F 9
  K D N 2
  K D F 6
K D P 1 0
---------
C A R E E R
```

PROBLEM 2

```
    O N E
    T W O
  F O U R
-----------
S E V E N
```

PROBLEM 3

```
U S A  U S A
F D R  N R A
------------
N R A  T A X
```

PROBLEM 4

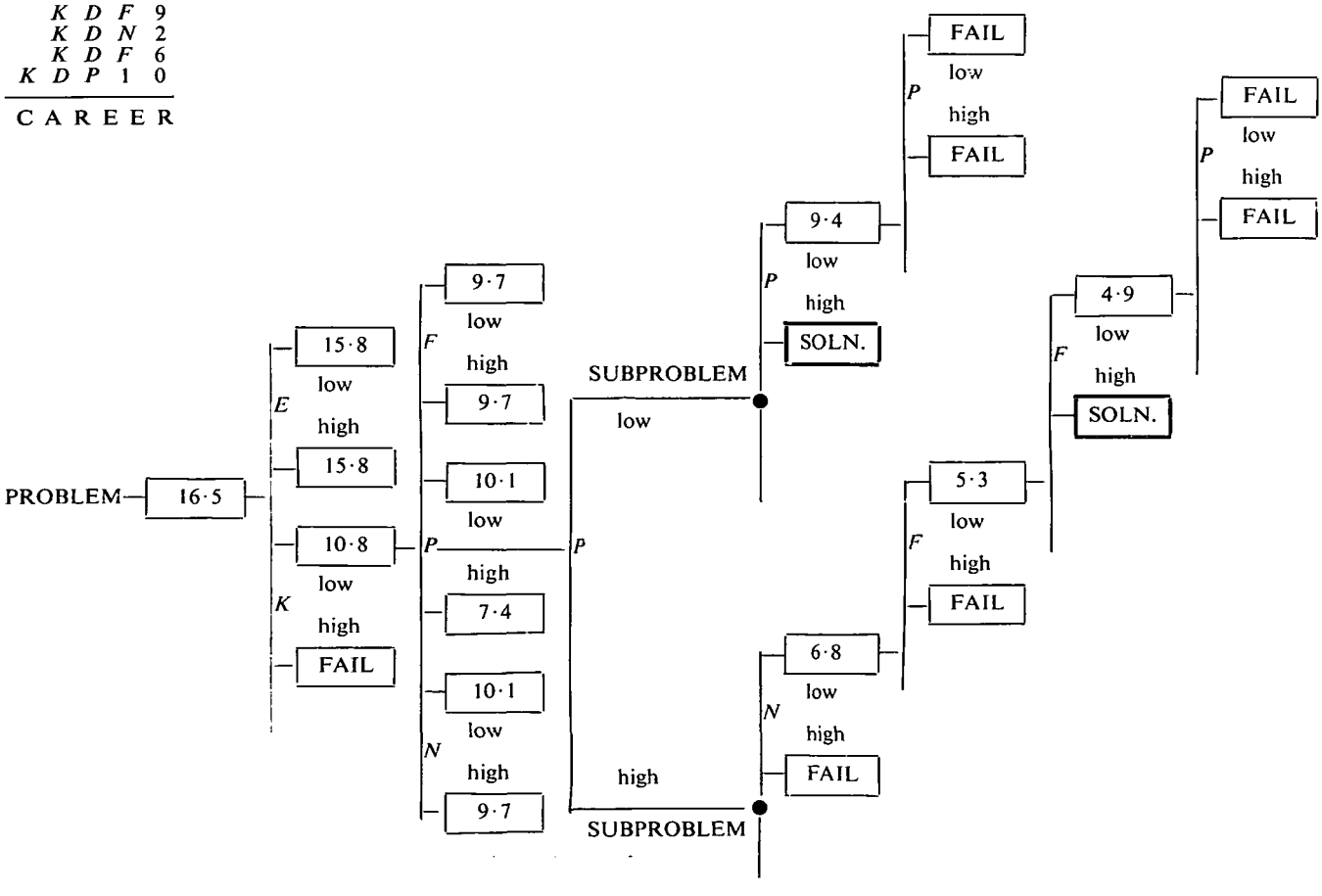
```
  H O C U S
  P O C U S
-----------
P R E S T O
```

PROBLEM 5

```
  S E N D
  M O R E
---------
M O N E Y
```

7. Concluding comments

This program gives a simple 'text-book' illustration of the use of heuristic tree search techniques (Golomb



The figures in the boxes give the 'space-size' after narrowing i.t. the product of ranges of variables on a natural log scale. FAIL indicates that a contradiction has been derived, SOLN indicates a solution. Variables were selected by the random approach.

Fig. 1. A typical search tree

and Baumert, 1965; Doran and Michie, 1966; Hart, Nilsson and Raphael, 1967; Burstall, 1967). It is closely related to the 'Branch and Bound' method of Little *et al.* (1963) and to the 'semantic tree' approach used in mechanical theorem proving (Robinson, 1968).

The main points in which it is more elaborate than the usual tree searching methods (such as the Doran and Michie 'Graph Traverser') are:

(i) Such methods usually take a fixed tree of hypotheses and concentrate on finding a good sequence of exploration. This program determines experimentally what tree of hypotheses shall be grown.

(ii) At each stage before any alternative hypotheses are formulated every effort is made to simplify the problem by making deductions. This produces a very much smaller tree.

Table 2. Results for word sum problems, various parameter values

Search was terminated on finding one solution
Mean of 5 replicates with different random numbers
R = random method
M = merit method

NO. OF VARIABLES TENTATIVELY SPLIT	TIME IN SECONDS											
	PROBLEM NO. (SEE TABLE 1)										MEAN	
	1		2		3		4		5			
	R	M	R	M	R	M	R	M	R	M	R	M
1	13	10	41	20	21	16	16	11	11	9	20	13
2	12	11	22	20	21	16	12	9	9	8	16	13
3	12	10	29	20	21	16	10	10	9	9	16	13

(The Elliott 503 has a time of 7·5 microseconds for most simple operations.)

Acknowledgements

This work was supported by a Kenward Memorial Fellowship at Birmingham University and a Science Research Council Fellowship at Edinburgh University. The material formed part of a Ph.D. thesis at Birmingham University and I am grateful to the Department of Engineering Production there for permission to publish. Thanks are also due to Professor D. Michie of Edinburgh University for his encouragement in the later stages of this work.

References

BEALE, E. M. L. (1965). Survey of integer linear programming, *Op. Res. Quart.*, Vol. 16 (2), pp. 219–228.

BROOKES, M. (1964). 150 *puzzles in cryptarithmetic*, New York: Dover.

BURSTALL, R. M. (1965). A tree searching method for solving integer linear inequalities, *Experimental Programming Reports: No.* 10, Edinburgh: Department of Machine Intelligence and Perception, University of Edinburgh. (Available on request.)

BURSTALL, R. M. (1967). Tree searching methods with an application to a network design problem, *Machine Intelligence* 1 (eds. N. L. Collins and D. Michie), Edinburgh and London: Oliver and Boyd, pp. 65–87.

DORAN, J. E., and MICHIE, D. (1966). Experiments with the Graph Traverser program, *Proc. R. Soc. A*, Vol. 294, pp. 235–259.

GOLOMB, S. W., and BAUMERT, L. D. (1965). Backtrack programming. *J. Ass. Comp. Mach.*, Vol. 12, pp. 516–524.

GOMORY, R. E. (1963). An algorithm for integer solutions to linear programs, *Recent Advances in Mathematical Programming*, (eds. R. L. Graves and P. Wolfe), McGraw-Hill, pp. 269–302.

HART, P. E., NILSSON, N. J., and RAPHAEL, B. (1967). A formal basis for the heuristic determination of minimum cost paths. *Research Report*, Stanford Research Institute, Menlo Park, California.

LITTLE, J. D. C., MURTY, K. G., SWEENEY, D. W., and KAREL, C. (1963). An algorithm for the travelling salesman problem, *Op. Res. Quart.*, Vol. 11, pp. 972–989.

ROBINSON, J. A. (1968). The generalised resolution principle. *Machine Intelligence* 3 (ed. D. Michie), Edinburgh: University Press, pp. 235–259.