An algorithmic search procedure for program generation

by M. H. HALSTEAD, G. T. UBER,
and K. R. GIELOW
Lockheed Missiles and Space Company
Sunnyvale, California

INTRODUCTION
A programmer in writing and checking out an arithmetic computer program provides both a procedural description of the program and a set of numeric test cases. Similarly, a student of elementary physics is often given a set of formulae to be used, and correct answers to problems. These two inputs in both cases provide a redundancy which gives some measure of confidence that the procedure is correct. In theory either is adequate, subject to the limitations of induction. While compilers operate upon the first type of data, several programs have been written which use the second type.

Friedberg¹,² in 1957 attempted to generate programs to compute Boolean functions in his experiments on machine learning. The impression which his experiments left was that a technique which generated and tested programs showed little promise. More recently Simon tackled the problem of generating IPL-V programs with his Heuristic Compiler,³ which uses means-end analysis of the before and after states of IPL-V stacks.

The procedure described in this paper searches for programs which compute arithmetic functions. Its input is one or more data sets each composed of one or more numeric arguments called parameters and a numeric function value. The exhaustive forward directed search—named the “British Museum Algorithm” by Newell, Shaw, and Simon⁴—was chosen because of its simplicity and because of the facility of computers for executing arithmetic programs. The search is optimized through utilization of syntactic symmetry as defined by Gelernter⁵ to eliminate redundant programs. Where applicable, the physical dimensions of the data are also used.

The search program described in the remainder of this paper establishes a benchmark for the exhaustive search approach. A basic search rate of 7,000 trial programs per second is achieved, which is a hundredfold improvement over Friedberg’s results. The program represents an application of computers to that oft forgotten subject in artificial intelligence: the numeric problem.

An experimental search program
A computer program named MAGIC has been written for the IBM 7094 to execute the search. For speed the program is written in NELIAC and FAP rather than in an interpretive language such as IPL-V. It now consists of some 16,000 words of instructions and tables.

Program generation
The goal of the search program is to find the shortest sequence of program steps which computes a function to within a specified tolerance. A program step is an operator-operand pair such as (ADD A). All possible sequences of program steps are generated and executed, and the results are tested against the function value. To ensure finding the shortest program, the search program first generates all one-step programs...

Figure 1—Partial program tree for three steps

From the collection of the Computer History Museum (www.computerhistory.org)
Tree searching

The program space may be considered a tree in which each program is represented by a path from the root to a leaf, or terminal node. (See Figure 1.) The paths are generated in Iverson's left list matrix order so that intermediate results may be saved and used without recomputation. The time required to execute the second program (CLA A, ADD A, MYP A) is thus the time needed to retrieve the result of the second step and to execute the new third step MPY A.

Successor selection

Successor criteria shorten the search by pruning subtrees rooted at each node. They are implemented with Successor Tables (Figure 2) which define the legal successors (marked “1”) to each operation. While these tables bear a superficial resemblance to CO-NO tables as employed in table-driven compilers, they are inherently quite different. Operations (marked “0”) which cancel previous operations (e.g., SUBTRACT A and ADD A) are prohibited. Operations marked “0” are suppressed because they result in computationally equivalent sequences due to commutativity (e.g., ADD A, ADD B versus ADD B, ADD A).

Operations violating certain physical constraints are eliminated also. Investigation of a sequence is terminated if floating point overflow or underflow occurs or if division by zero is attempted. The search is also terminated if the choice of operation results

Note: The operands A, B, C, D and E are parameters: T is a temporary storage cell, and q is the intermediate result corresponding to the contents of the IBM 7094 AC-register. The first seven operators correspond to the 7094 commands: Clear and Add, Clear and Subtract, Floating Add, Floating Subtract, Floating Multiply and Proceed, Floating Divide and Proceed, and Store. The next operator exchanges the AC-register with T, and the remaining operators replace A with the indicated function of q. On the first step only the 0+ and 0- operators may be executed.

Figure 2—Successor tables

<table>
<thead>
<tr>
<th>CURRENT OPERATION</th>
<th>SUCCESSING OPERATIONS</th>
<th>a &amp; T</th>
<th>a ≺ T</th>
<th>a ≻ T</th>
<th>a + T</th>
<th>a - T</th>
<th>a / T</th>
<th>a * T</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &amp; T</td>
<td>a ≺ T</td>
<td>a ≻ T</td>
<td>a + T</td>
<td>a - T</td>
<td>a / T</td>
<td>a * T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a + T</td>
<td>a ≺ T</td>
<td>a ≻ T</td>
<td>a + T</td>
<td>a - T</td>
<td>a / T</td>
<td>a * T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a ≺ T</td>
<td>a ≻ T</td>
<td>a + T</td>
<td>a - T</td>
<td>a / T</td>
<td>a * T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a ≻ T</td>
<td>a + T</td>
<td>a - T</td>
<td>a / T</td>
<td>a * T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a + T</td>
<td>a - T</td>
<td>a / T</td>
<td>a * T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a - T</td>
<td>a / T</td>
<td>a * T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a / T</td>
<td>a * T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a * T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and then successively generates longer programs until a successful program is found.

Table 2

<table>
<thead>
<tr>
<th>OPERATIONS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, D, E</td>
<td>Parameters</td>
</tr>
<tr>
<td>T</td>
<td>Temporary storage cell</td>
</tr>
<tr>
<td>q</td>
<td>Intermediate result</td>
</tr>
<tr>
<td>Figure 1</td>
<td>Successor Tables</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Successor selection</td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
in a dimensional inconsistency (e.g., apples may not be added to oranges). The dimensionality of the intermediate results is computed, and the operator choice is tested against the following rules:

1. For addition and subtraction the intermediate result and the parameter must have the same dimension.
2. For square root the dimensions must be perfect squares. *
3. For trigonometric functions, the intermediate result (i.e., the argument) must be dimensioned. Because there can be no successful program shorter than the shortest dimensionally correct program, only those operations which affect dimensions are tried until the first such program is found.

The successor tables are defined by macro instructions and are executed using a combination of double indirect addressing and triple indexing. The operators are also defined by macros and are expanded for each argument. Coding is included in the operator definitions which tests the dimensions and computes the dimensions of the result. The six dimensions are each represented as six-bit fields, the exponents having a range of ±15. The numerical values are stored as floating point numbers.

**Test routine**

When the last step of a sequence has been generated and executed, the result is checked against the desired result and the check counter incremented. If the dimensions agree and the numerical value is within a specified tolerance for the current function value, the successful sequence is saved in an encoded form for further processing.

**Timing**

Most of the search time is spent in processing the terminal nodes. It requires 60 microseconds to generate and execute the terminal step and 60 microseconds to check the final result and increment the check counter. The search is rapid enough so that 10 percent of the machine time is required merely to tally the number of sequences checked.

**Search length**

Given P parameters and R operators in the repertoire at each step, the maximum number of possible operator-parameter pairs is the product PR. With \( \beta \) pairs at each step, the number of possible sequences of length S is \( \beta^S \), or as a maximum (PR)\(^S\). With the British Museum Algorithm, sequences are examined in order of length. To find a sequence of length S requires Q checks where

\[
Q = \sum_{i=1}^{S} \beta^i
\]

Twenty operators and five parameters are sufficient for the problems being investigated; hence, \( \beta \) equals 60. Not all operators need operate upon all parameters. Let P be the number of input parameters, \( R_p \) the numbers of operators acting upon input parameters, T the number of temporary storage cells, \( R_T \) the number of operators acting upon such cells, and F the number of operators (Functions) acting upon intermediate results. We now define

\[
\beta = PR_p + TR_T + F
\]

For the present Successor Tables: P is 5, \( R_p \) is 12; T is 1; \( R_T \) is 6, and F is 4. Hence \( \beta \) is 40, yielding the curve "\( \beta=40 \)" in Figure 3. The other constraints in the Successor Tables reduce the value of \( \beta \) still further. For undimensioned parameters (NO DIMENSIONS) the effective value is 18. When the parameters are dimensioned such that addition and subtraction are completely suppressed (COMPLETE DIMENSIONS), the value drops to 3. However, for a typically dimensioned problem \( \beta \) has a value of about 6. The curves are computed for five parameters, and the points in the figure are labeled with the search time in seconds.

**Examples**

The search timing for most problems can be predicted from Figure 3. The examples below indicate typical results and also illustrate additional features of the system.

Multiple data sets provide a better basis for generalization than do single data sets.* To avoid redundant printout, programs are listed only if they satisfy all data sets or else satisfy a combination of data sets not included in combinations satisfied by previously listed programs.

The results of a search over multiple data sets are expressed by the list of programs and by a Boolean cover matrix C in which each element \( c_{p,d} \) equals 1 if the program p computes an acceptable value for the data set d, and equals 0 otherwise. The decision

---

*This constraint was introduced because the current representation of dimensions does not permit fractional powers.

---

*Rigg's advice to biologists seems appropriate: "You should check any procedure about which you are dubious by substituting sensible numerical values for the algebraic terms and making sure that the procedure is correct for these numbers. To guard against the possibility that a wrong procedure may by chance give a correct result with the particular numbers chosen, you should check by this method of numerical substitution twice with two different sets of values."
program generator, operating upon C and the data set parameters, searches for a minimal set of programs \( p \) each with an associated Boolean function \( b_p \) such that:

1. For each data set \( d \) there exists a program \( p \) such that \( b_p(d) = 1 \)
2. \( b_p(d) \) implies \( c_{p,d} \)
3. \( b_p \) is a Boolean expression of threshold functions of single parameters

**Example one: approximating functions**

The tolerance was set to 10 percent so that approximate solutions would be listed. The search, using input shown in Table I, resulted in the programs of Table II and required 3.1 seconds.

The expression \((1/\sin P)^{1/2} = 1.09\) is in effect a coefficient used to “fit” the function to the first three data sets. Coefficients (for example, the parameters \( v_o \) and \( T_o \)) must usually be supplied while the functions are generated, in contrast to most curve fitting programs for which the functions are supplied and the coefficients are generated.

**Table I**

<table>
<thead>
<tr>
<th>Program</th>
<th>Name Dimension</th>
<th>Parameter</th>
<th>( v_o ) (m/sec)</th>
<th>( T_o ) (deg)</th>
<th>( v ) (m/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data Set</td>
<td>( P ) (deg)</td>
<td>( T ) (deg)</td>
<td>( v_o ) (m/sec)</td>
<td>( T_o ) (deg)</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>0</td>
<td>331.7</td>
<td>273</td>
<td>331.7</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>20</td>
<td>331.7</td>
<td>273</td>
<td>344.0</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>100</td>
<td>331.7</td>
<td>273</td>
<td>386.0</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
<td>500</td>
<td>331.7</td>
<td>273</td>
<td>553.0</td>
</tr>
<tr>
<td>5</td>
<td>1.0</td>
<td>1000</td>
<td>331.7</td>
<td>273</td>
<td>700.0</td>
</tr>
</tbody>
</table>

**Figure 3 — Search times**

**Example two: decision program**

This shows a piecewise fit to a payroll function. The input data for two employees are given in Table III. The search, using a tolerance of 1 percent, found 780 programs of which the 8 in Table IV were listed.

**Table II**

<table>
<thead>
<tr>
<th>Program</th>
<th>Function</th>
<th>Checks</th>
<th>Cover Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( V = V_0 )</td>
<td>5</td>
<td>1 1 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>( V = (1/\sin P)^{1/2} V_0 )</td>
<td>7,464</td>
<td>1 1 1 0 0</td>
</tr>
<tr>
<td>3</td>
<td>( V = (T + T_0)/T_0^{1/2} V_o )</td>
<td>16,663</td>
<td>1 1 1 1 1</td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
During 3.26 minutes of computer time, 1,311,126 programs were tested.

The decision program generation required 0.02 minutes and 68 trials. The generator combined three programs (in NELIAC) as shown in Figure 4. The composite program has 16 steps although the maximum search length was five steps.

**Example three: program library**

This example illustrates the construction of a specialized subprogram library. For each of the six blackbody radiation problems, the resulting programs in Table V were stored for use on the remaining problems. A stored subprogram can be executed as a step, its input being a permutation of the input parameters and the contents of $T$. The program for Function 5, for example, consists of

1. Function 4(A, B, C, D)
2. DVP E
3. DVP E

where parameters A, B, C, D and E correspond to the variables $r, e, s, T$, and $d$, respectively. While the maximum search time for a given number of steps is roughly proportional to library size, a properly specialized library could significantly shorten the search depth.

**DISCUSSION**

The principal object of this paper has been to investigate the forward directed search as implemented on present-day computers. While improvements in computer performance would be expected to lower the search cost still further, the greatest reductions may be expected from the use of matching and selection procedures which extract operators, parameters, and subprograms most applicable to a given problem and supply them to the search procedure.

**REFERENCES**

1. R M FRIEDBERG

*A learning machine part 1*
HOURS WORKED = 0.0,
DAYS OF WEEK = 0.0,
HOURLY RATE = 0.0,
SHIFT LENGTH = 8.0,
TWO = 2.0;

COMPUTE DAYS PAY:

\{
\begin{align*}
\text{IF DAY OF WEEK} & = 7.0: \\
& (\text{HOURS WORKED} + \text{HOURS WORKED}) \times \text{HOURLY RATE} \rightarrow \text{DAYS PAY}; \\
\text{IF NOT,} & \\
\text{IF HOURS WORKED} & \leq 8.0: \\
& \text{HOURS WORKED} \times \text{HOURLY RATE} \rightarrow \text{DAYS PAY}; \\
\text{IF NOT,} & \\
& \left(\frac{\text{HOURS WORKED} - \text{SHIFT LENGTH}}{\text{TWO}} + \text{HOURS WORKED}\right) \times \text{HOURLY RATE} \rightarrow \text{DAYS PAY}; \\
\end{align*}
\}

Figure 4—Output for decision program

Table V

<table>
<thead>
<tr>
<th>Function</th>
<th>Steps</th>
<th>Checks</th>
<th>Total Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 sT^4</td>
<td>5</td>
<td>94</td>
<td>5</td>
</tr>
<tr>
<td>2 est^4</td>
<td>2</td>
<td>58</td>
<td>6</td>
</tr>
<tr>
<td>3 AsT^4</td>
<td>2</td>
<td>237</td>
<td>7</td>
</tr>
<tr>
<td>4 r^2 est^4</td>
<td>2</td>
<td>1259</td>
<td>8</td>
</tr>
<tr>
<td>5 r^2 est^4/d^2</td>
<td>3</td>
<td>17296</td>
<td>10</td>
</tr>
<tr>
<td>6 AsT^4/d^2π</td>
<td>4</td>
<td>155789</td>
<td>9</td>
</tr>
</tbody>
</table>

IBM Journal of Research and Development vol 2 pp 2-13
Jan 1958

2 R M FRIEDBERG B DUNHAM J H NORTH
A learning machine part II
IBM Journal of Research and Development vol 3 pp282-7
Jul 1959

3 HERBERT A SIMON
The heuristic compiler
Memorandum RM-3588-PR The RAND Corporation
Santa Monica Calif 1963

4 A NEWELL J C SHAW H A SIMON
Empirical explorations of the logic theory machine
Proceedings of the Western Joint Computer Conference
pp 218-39 1957
Reprinted in Feigenbaum E A and J Feldman
Computers and thought
New York 1963

5 H GELERNTER
A note on syntactic symmetry and the manipulation of formal systems by machine
Information and control
vol 2 pp 80-89 Apr 1959

6 K E IVESON
A programming language
John Wiley & Sons Inc New York 1962

7 M H HALSTEAD
Machine-independent computer programming
Spartan Books Washington D C 1962

8 D S RIGGS
The mathematical approach to physiological problems
Williams & Wilkins Baltimore p 3 1963