1974

Experimental Validation of a Structural Property of Fortran Algorithms

Necdet Bulnut

Maurice H. Halstead

Rudolf Bayer

Report Number:
74-115

EXPERIMENTAL VALIDATION OF A STRUCTURAL PROPERTY OF FORTRAN ALGORITHMS

Necdet Bulut
M. H. Halstead
Rudolf Bayer

April 1974

CSD TR 115

Experimental Validation of a Structural Property of Fortran Algorithms

Necdet Bulut*
M. H. Halstead**
Rudolf Bayer***

Recent studies [1,3,8,9] have shown that theory predicts, and limited experimental data confirm, the existence of a functional relationship between the length, N, of the expression of an algorithm and the number of distinct operators, η_1, and of distinct operands, η_2, required to express that algorithm in various languages. Data previously examined, however, was restricted to a small sample of rather short, published algorithms, for which η_1, η_2 and N could be measured manually.

In order to guarantee that identical, clerical-error-free counting or measuring methods are applied to a sample of programs which are both greater in number and longer in length, it was necessary to reduce the counting procedure itself to an algorithm. The automatic counting procedure was then applied to each of the 429 Fortran programs in the FORTOPL library of the Purdue University Computing Center. At the time that the experiment was performed, that library contained an additional 76 decks which were mixed Fortran-assembly language programs and 11 common decks which contained interspersed system modification information, none of which could be used in the analysis.

The sizes of the 429 programs in the sample extended from three or four statements for the smallest to 1017, 1140 and 1674 Fortran statements for the three largest.

---

*Purdue University and Middle East Technical University, Ankara, Turkey
**Purdue University
***Technische Universitat, Munich, Germany

The counting algorithm provided, implicitly, the definitions of distinct operators, η_1, distinct operands, η_2, total operator usage, N_1, and algorithm length, $N = N_1 + N_2$. (These definitions are described later). Explicitly, it provided tabulations of each of these five parameters for each of the programs processed. The relationship:

$$N \doteq \eta_1 \log_2 \eta_1 \quad \eta_2 \log_2 \eta_2 \tag{1}$$

was then examined by plotting the left hand side of the relation as N(observed) against the right hand side as N(calculated). The machine plot of all data points is shown in figure 1, which also displays the statistical linear regression line ($N_0 = .94N_c + 125.27$; corr. coeff: .95). In order to provide resolution for the large number of points near the origin, the scale of the first plot was then expanded by a factor of 20, and the result shown in figure 2.

In our opinion this experiment confirms the previously reported existence of a functional relationship between η_1, η_2 and program length, and suggests that equation 1 can be used as a fair approximation to it.

This result can be seen to have meaningful implications in software physics, when it is noted that these five parameters are the same as those used earlier [1,59] to define algorithm volume, V, and estimate algorithm level, L, as

$$V = (N_1 + N_2)\log_2(\eta_1 + \eta_2) \tag{2}$$

$$L \doteq \frac{2}{\eta_1}\frac{\eta_2}{N_2} \tag{3}$$

from which it was shown [9] that for any given algorithm, it appears that the product LV is invariant under translation, and depends only upon the number of input-output variables. Further, they are the same parameters used to estimate the time required to program a preconceived, one module algorithm in a language known to the programmer [2,4,6,7] from the relation

$$T = \frac{V}{LS} = \frac{1}{3000} \frac{\eta_1}{\eta_2} N_2 (N_1 + N_2) \log_2(\eta_1 + \eta_2) \quad . \quad (4)$$

Consequently, it is of interest to examine, even for a single language, the precise definitions of η_1, η_2, N_1 and N_2 implied by the processing algorithm.

A Counting Algorithm for Fortran Programs

Basically, the counting algorithm includes a lexical analyzer and a parser, similar to those which would be found in a Fotran compiler. It recognizes symbols, constants, variable names and keywords, and parses the Fortran statements. It follows the basic principles that declaration statements are not part of the pure algorithm, that only variables and constants are operands, and that any symbol or positional notation which may have an effect upon an operand is an operator.

Specifically, it embodies the following rules:

a) Material extraneous to the pure algorithm, (i.e. comments, specification statements, input/output statements, STOP RETURN and END) are ignored.

b) All arithmetic, boolean, and replacement operators are counted.

c) Function names are counted as operators.

d) A GOTO operation is completed by a label attached to it. Since the label is not an operand, the combination GO TO L1 is one operator, hence, GO TO L2 is a different operator. Also, labels in a computed GO TO statement are counted as distinct GO TO L_i provided that the L_i are distinct.

e) An IF statement is counted as an operator.

f) Statement labels following an arithmetic IF are counted as GO TO L_1, GO TO L_2, and GO TO L_3.

g) An ASSIGN statement for an assigned GOTO is counted as a replacement operator.

h) Any occurrence of a pair of grouping symbols is counted as a single operation of the grouping operator. In addition to parenthetical grouping this includes the case in which a DO with a statement number is the first element of a pair, and the label attached to the range statement, (or label and CONTINUE, if it exists) is the last element of the pair.

i) Parenthesis pairs denoting subscription are counted as single subscription operators.

j) A comma is counted as an operator wherever it occurs.

k) Positional notation denoting the start of a new statement is counted as an EOS operator.

l) Unless they are involved in a special impurity class [10], to which rules m through o apply, all variables and constants are counted as operands.

m) Formal parameters are not considered unique, because they are synonomous with actual parameters which are recognized as distinct in the calling program. Their occurrence contributes to η_2 but not to η_2.

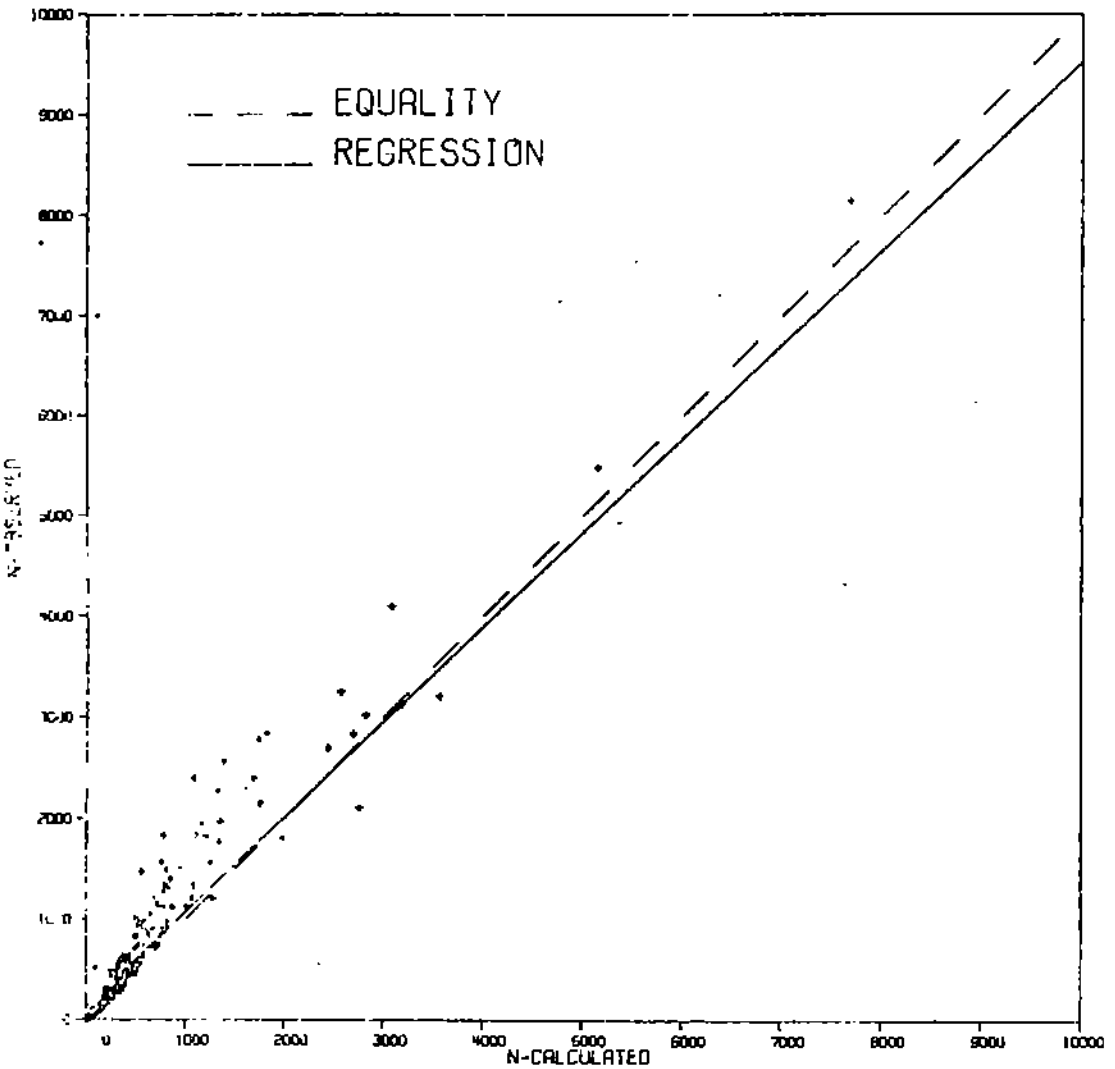
n) Repeated uses of a given array variable with the same index are considered to be common subexpressions, and counted as occurrences of a temporary variable replacing the common subexpression.

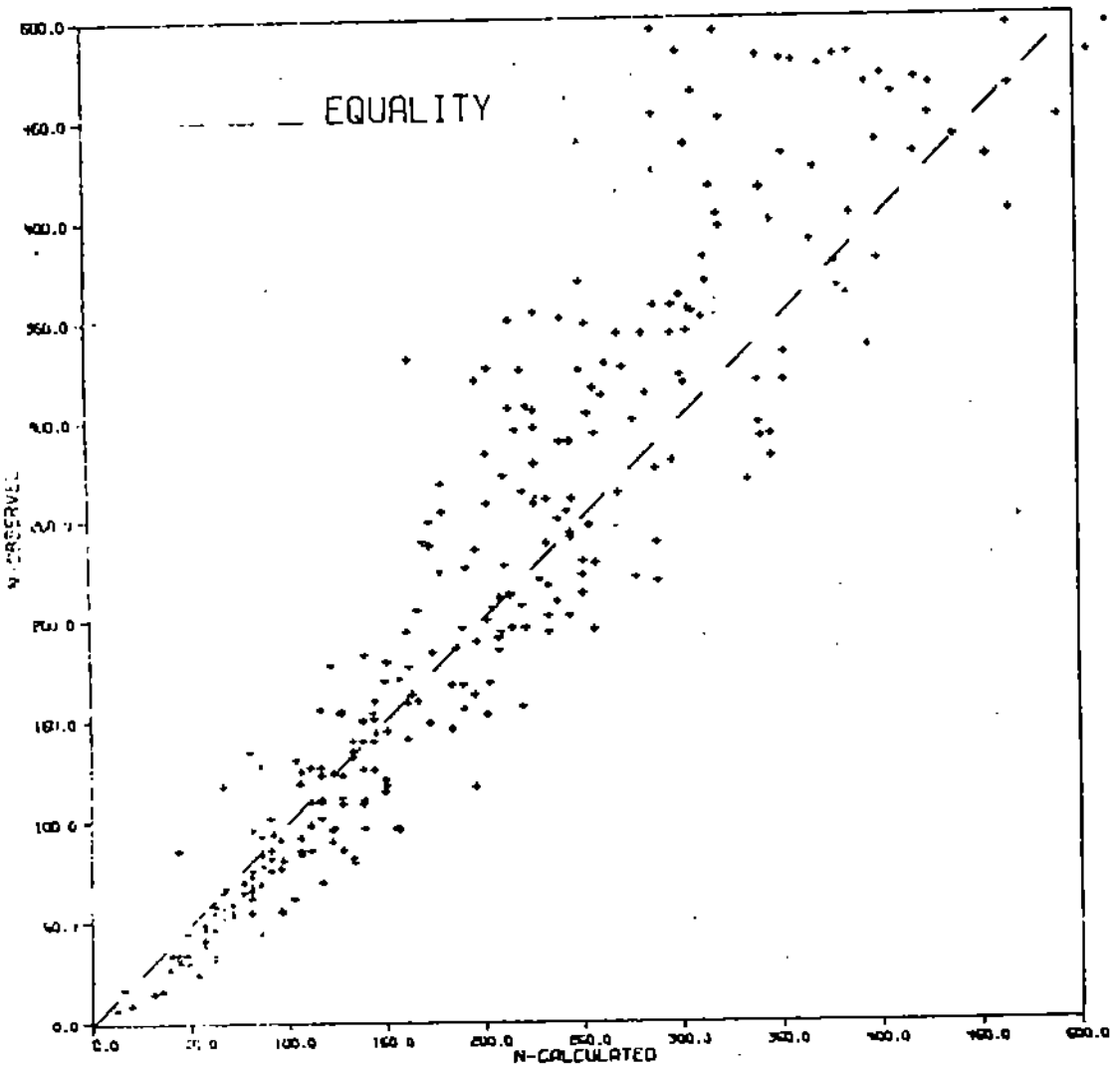
o) Ambiguity in the use of local variables which have the same names in different subprograms is resolved by the counting algorithm.

The first 12 of these 15 rules are sufficiently general that they may be taken as axiomatic or intuitively obvious. The last three, on the other hand, result from the fact that the Fortran language does not provide suitable mechanisms by which the programmer can avoid all occurrences

of common subexpressions, ambiguous operand usage, and synonomous operand usage.

In summary, the experiment described, which represents more than an order of magnitude increase in both the number of programs tested and in the range of program lengths previously examined, confirms the existence of a functional relationship between the measurable parameter η_1, η_2 and N. It should be noted however, that while equation 1 may serve as a useful approximation to that functional relationships over the conditions tested, it is still only an approximation.


EQUALITY
REGRESSION
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
N-CALCULATED


EQUALITY
N-CALCULATED

REFERENCES:

[1] Halstead, M.H. : Natural Laws Controlling Algorithm Structure?, ACM-SIGPLAN Notices, Vol. 7, No. 2, Feb. 1972, pp. 19-26.

[2] Halstead, M.H. : A Theoretical Relationship between Mental Work and Machine Lagnuage Programming, CSD TR 67, Purdue University, Dept. of Computer Sciences, Feb. 1972.

[3] Bayer, R. : A Theoretical Study of Halstead's Software Phenomenon, CSD TR 69, Purdue University, Dept. of Computer Sciences, May 1972.

[4] Halstead, M.H. and Bayer, R. : Algorithm Dynamics, CSD TR 72, Purdue University, Dept. of Computer Sciences, May 1972.

[5] Halstead, M.H. : Language Level, a Missing Concept in Information Theory, ACM-SIGME Performance Evaluation Vol. 2, No. 1, March 1973, pp. 7-9.

[6] Zislis, P.M. : An Experiment in Algorithm Implementation, CSD TR 96, Purdue University, Dept. of Computer Sciences, June 1973.

[7] Halstead, M.H. and Zislis, P.M. : Experimental Verification of Two Theorems of Software Physics, CSD TR 97, Purdue University, Dept. of Computer Sciences, June 1973.

[8] Halstead, M.H. and Bayer, R. : Algorithm Dynamics, Proceedings of the ACM Annual Conference 1973, pp. 125-135, Atlanta.

[9] Bulut, N. : Invariant Properties of Algorithms, Ph.D. Thesis, Purdue University, Dept. of Computer Sciences, Aug. 1973.

[10] Bulut, N. and Halstead, M.H. : Impurities Found in Algorithm Implementations, ACM-SIGPLAN Notices, Vol. 9, No. 3, March 1974, pp. 9-12.