MEASUREMENTS OF PROGRAM SIMILARITY IN IDENTICAL TASK ENVIRONMENTS

H.L. Berghel
Department of Computer Science
University of Nebraska
Lincoln, NE 68588

D.L. Sallach
Computer Information Management
College of St. Mary
Omaha, NE 68124

ABSTRACT: This paper summarizes the results of a study which compared the efficiency of two methods of measuring program similarity in the context of novice programmers trying to reach identical objectives. Both methods look for similarity by comparing 'program profiles'. Such profiles are created by feature extraction routines which map each program onto a tuple $\langle f_1, f_2, \ldots, f_n \rangle$ where each f_i is a count of an occurrence of a particular feature. A comparison routine is then invoked which detects similarities between tuples. The results showed that in this environment the comparison routine based on the Halstead metric failed to perform as well as a conceptually simpler alternative.

INTRODUCTION:

The present study of program similarity arose out of a practical need. The first author of this paper became interested in automated plagiarism detection systems while he was the supervisor of a computer literacy program serving a Business College [1]. Typically, the grading staff handled 4000 programs per term (500 students submitting eight programs each). On an average, less than 1% of the programs were found to be candidates for plagiarism. Since the staff had independent knowledge that close collaboration far exceeded this percentage, an automated system was called for.

Over several semesters, a wide range of features which had been identified as possible indices of program similarity were empirically evaluated by the staff of graders. Through this informal process the original list of approximately 30 possible features was pruned to 15 key features which seemed to be most useful in identifying program similarity in this environment.

The body of programs which serve as the data for analysis are drawn from summer session classes. The summer classes are smaller, which reduced the number of programs to be manually compared to a manageable level. In all, approximately 700 FORTRAN programs (100 students at 7 assignments) were studied, approximately 55% of which were included in the pretest.

We had determined during the previous academic year that the following features were the most useful in distinguishing programs in this environment:

f_1 - code lines
f_2 - total lines
f_3 - continuation statements
f_4 - keywords
f_5 - real variables
f_6 - integer variables
f_7 - total variables
f_8 - assignment statements (initialization)
f_9 - assignment statements
f_{10} - declared reals
f_{11} - declared integers
f_{12} - total operators (Halstead N_1)
f_{13} - total operands (Halstead N_2)
f_{14} - unique operators (Halstead n_1)
f_{15} - unique operands (Halstead n_2) .

It should be noted that the last four features form what has been referred to in the literature as a Halstead metric (see below). It is interesting to note that these features did not appear distinctively stronger than other features on the list.

Establishing the short list of features did not eliminate the difficulties inherent in plagiarism detection. Features appeared to have differential utility based upon the nature of the assignment. It seemed likely that features would exhibit clustering, and the identification of such patterns was a non-trivial task. Aside from the Halstead features, no theoretical basis was available to provide a framework with which to interpret feature effectiveness. Nor was the relationship of the Halstead metric to the remaining features self-evident. Since the immediate goal was to define one or more metrics as effective in plagiarism detection, it was determined to perform a factor analysis to reduce the complexity of feature interaction, and to identify a limited number of common factors with which to proceed.

Factor analysis employs a set of observed variables to identify the one or more unobserved variables which are postulated by the factor analysis model [18]. In the present study, the 15 features are the observed variables, and they are used to identify emergent factors which can account for a high percentage of the variance. The data which was factor analyzed was collected during first summer session, for the class described above. Data collected for an equivalent class section in the second summer session was used to test the application of the two program similarity metrics which emerged from the factor analysis (see next section).

It was determined to employ a quartimax orthogonal rotation in order to maximize the independence of the underlying factors. The first factor accounted most of the variance for the pooled data, for each of five programming assignments. Specifically, the first factor accounted for no less than 60% and, on one occasion, over 90% of the variance. For the pooled data, 74.9% of the variance is explained by the first factor. Inasmuch as these figures are quite robust (and comparable second factor loadings were quite weak), it was determined to focus upon the first factor loadings in constructing the resulting metric(s). Table 1 summarizes the factor loadings for each of the analyzed features.

Initially, the pooled data was used to identify the items for each factor. However, as Table 1 indicates, the items which loaded high on the pooled data were not necessarily consistent items for each factor type, due to differences in rotation effect. Because generality was preferable to assignment specific indicators, it was determined to establish the generality of each item by calculating a mean factor rank for each assignment. Specifically, each item with a first factor loading of .5 or higher on the pooled data was ranked as to its first factor loading for each of five assignments. The results, including the grand mean for each factor, are shown in Table 2.

When all assignments were taken into account, the three strongest factors were three components of the Halstead metric; Total Operators was far and away the strongest single item. The fourth Halstead feature (Unique Operators) finished eighth in composite rank, behind a seemingly ad hoc cluster of features generated according to empirical criteria. The cutoff point for inclusion was made after the first eight items on the mean rank, based on three criteria. First, this point allowed inclusion of all four Halstead parameters. Second, the subsequent items were more erratic and considerably weaker. The strongest of the excluded variables was more than two standard deviations from the aggregate mean rank, and nearly one standard deviation lower in rank than Unique Operators. Third, the excluded variables were either more language specific than the retained items, or redundant. In either case, they could add little to the development of a general metric.

The eight items emerging from the factor analysis were split to form two separate classes for use in the actual plagiarism detection tests. The four Halstead features were considered as a single metric to maintain theoretical continuity. The four non-Halstead items were grouped to form a separate metric. The policy of treating the four as an integrated metric is supported by the fact that they formed a contiguous group in the mean rank results. Consideration of the theoretical basis for the non-Halstead metric will be deferred to a later section.

FINDINGS:

The actual plagiarism detection routine had as its universe a group of students taking the same course as the one on which the pretest was defined, during the second summer session. The program which tested the two metrics directly compared the two four-featured metrics. Following Ottenstein [21], we employed a method of cumulative satisfaction. Two programs were adjudged similar if and only if each of the values within one profile were within a certain range of the corresponding value of the other. The tightness of the mesh of the detection sieve is thus tuned by varying the values of the comparison tuple $C = \langle c_1, c_2, c_3, c_4 \rangle$ so that any two program profiles $P_1 = \langle f_1, f_2, f_3, f_4 \rangle$ and $P_2 = \langle f_1', f_2', f_3', f_4' \rangle$ are said to be similar if and only if for all i: $|f_i^1 - f_i^2| \leq c_i$. Should the comparison fail for any feature - pair, it is said to fail for the entire tuple - pair.

Once the two metrics had been calculated, the validity of their profiles was estimated by comparison with the independent judgement of the graders. The general result of these comparisons is that the Halstead metric consistently detected similarities which did not exist. The alternative metric, in contrast, showed itselt to be consistently more reliable.

To illustrate, Table 3 lists the numbers of program pairs which were found to be similar for the $C = \langle 0, 0, 0, 0 \rangle$ and $C = \langle 1, 1, 1, 1 \rangle$ cases by each method. Every pair of programs so detected was then manually reviewed and assessed. The '% correct' figure represents that percentage of the detected pairs for which the graders and present author felt a case for plagiarism might reasonably be made. On the other hand, while the alternative method was too narrow, it was entirely accurate within its limited scope. Appendix I illustrates how the Halstead method mistakenly matched one program pair. Though there is little in common between these sections of code (given that both parent programs accomplished the same task) their Halstead profiles are identical.

Understandably, the problems of excessive breadth is exacerbated as one increases the values within the comparison tuple. Appendices II and III contain two pairs of programs that were found to be nearly identical at $C = \langle 1, 1, 1, 1 \rangle$ by the Halstead method. Again, given that these program pairs have identical objectives and were written by novices at the same stage of their programming education, they are notable for their dissimilarity. In fact, the program in Appendix IIA contains a logic error which prevents it from running correctly. While the alternative method was also too broad at $C = \langle 1, 1, 1, 1 \rangle$ the dissimilarities between programs mistakenly identified as similar were less radical than for those identified by the Halstead Method. In short, even when the Halstead Method was accurate, it was frequently accurate for the wrong reasons.

CONCLUSION:

Plagiarism detection systems may be inaccurate in one of two ways. The system may be too broad (i.e., 'detect' similarities which don't exist) or too narrow (i.e., fail to detect similarities which do exist). When working with student programming assignments where each program has the same objective, inaccuracies are usually of the former type. As the results below reveal, one major problem with the Halstead Profile detection system is that it is consistently too broad. In terms of our simile, the mesh of the Halstead Profile detection sieve could not be made fine enough. A broader issue concerns why the Halstead metric was ineffective and, more specifically, why the alternative metric was able to achieve relative success.

From a theoretical standpoint, the alternative metric (AM) was both broader and narrower than the Halstead metric (HM). Let us compare the two metrics on a feature by feature basis. Code Lines is one feature of AM which is not present in HM. Although there is no direct analog, Code Lines may generally be considred a more coarse measure than any of the HM feature. The two HM features which have no direct counterpart in AM are Unique Operators and Unique Operands. They are finer than any of the AM features. Thus, a comparison of features which are present in one metric, and absent from the other, finds AM to be significantly broader than HM.

A second feature of AM is Assignment Statements, which is a subset of Total Operators in HM. Accordingly, the number of Assignment Statements is a more specific, or narrower, measure than its HM counterpart. A third possible comparison is between Total Variables in AM and Total Operands in HM. Here, too, the relationship is clear, The latter includes constants, whereas the former does not. Thus, the comparable AM feature is, again, narrower than the corresponding HM feature.

The relationship between Keywords in AM and Total Operators in HM is more complex than the preceding cases. The two features overlap and, thus, each is broader <u>and</u> narrower than the other in some respects. The lists in Table 4 provide a basis for assessing the comparative impact of each measure. It can be seen that, while the two measures overlap to a great extent, each includes items which have been omitted by the other. It is also clear that AM includes many more items, and will thus be generally broader.

Overall, the Alternative Metric contains two features which are broader than their Halstead counterparts, and two that are narrower. It is therefore not surprising that the two metrics would yield different results. It is noteworthy, however, that the Alternative Metric did provide greater discrimination and served as a more effective detection sieve. Only if most programs fell into the narrow range where the Halstead features are focussed could that metric provide superior identification of program similarity.

For software science, the broader issue concerns the implications which the study has for the development of a well-defined theory of program structure. Our factor analysis demonstrates that a number of features, of which Halstead's compose only a part, appear to lie along the same underlying dimension. When applied to the practical problem of program similarity identification, features were seen to have differential utility based upon situational factors such as assignment. We believe that if such situational problems were varied more fundamentally (e.g.,accross different program languages and by different levels of users), the utility of specific features would vary even more widely.

We are forced to conclude that there is nothing unique about the features isolated by the Halstead metric. While the application of quantitative methods to program structure has shown itself to be productive, the Halstead features seem to have no unique theoretical or practical properties which make them singularly effective indicators of program structure. More specific features (such as a count of the frequency of a specific operators like assignment statements) and more general features (such as total code lines, or perhaps even the size of an object module) may be equally or more effective than the components of the Halstead metric. At the very least, the isolation of the most powerful indicators of program structure is a task which is as yet incomplete. More fundamentally, perhaps contextual factors will prevent <u>any</u> set of features from achieving this type of conceptual primacy.

AFTERWORD:

A great deal has been written about attempts to identify measurable properties of programs [3,4,5,6,8,10,19]. Almost all such studies deal either directly or indirectly with Halstead's pioneering work in software science [12-17], an excellent overview of which appears in Fitzsimmons and Love [9]. Ottenstein [20,21] was the first to extend Halstead's work to the topic of program plagiarism. Alternative plagiarism detection systems (i.e. those which do not use the Halstead metric) have been proposed by Donaldson, et al, [7] and Grier [11]. The results summarized here are expanded and placed in a larger perspective in [2].

REFERENCES:

[1] Berghel, H. and C. Daly: "A Comparison of Three Approaches Toward Teaching Computer Literacy", Proceedings of the IEEE ED COMPCON-83, IEEE Computer Society, Spring, 1984 (forthcoming).

[2] Berghel, H. and D. Sallach: "Identifying Program Similarity: the Limits of the Halstead Metric" (forthcoming).

[3] Bulut, N.: "Invariant Properties in Algorithms", Ph.D. Thesis, Purdue University, (1973).

[4] Bulut, N. and M. Halstead: "Impurities Found in Algorithm Implementations", SIGPLAN Notices, (March 1974), pp. 9-12.

[5] Bulut N., M. Halstead, and R. Bayer: "Experimental Validation of a Structured Property of FORTRAN Algorithms", Computer Science Department Internal Report CSD-TR 115, Purdue University, (April 1974).

[6] Cornell, L. and M. Halstead, "Predicting the Number of Bugs Expected in a Program Module", Computer Science Department Internal Report CSD-TR 205, Purdue University, (October, 1976).

[7] Donaldson, John L., Ann-Marie Lancaster and Paula H. Sposato: "A Plagiarism Detection System", SIGSCE Bulletin, Vol. 13, No. 1, (February 1981), pp. 21-25.

[8] Elshoff, H., "Measuring Commercial PL/1 Programs Using Halstead"s Criteria", SIGPLAN Notices, May, 1976.

[9] Fitzsimmons, A. and T. Love: "A Review and Evaluation of Software Science", Computing Surveys, Vol. 10, No. 1, (March 1978), pp. 3-18.

[10] Gordon, R. and M. Halstead, "An Experiment Comparing FORTRAN Programming Time with the Software Physics Hypothesis", Proceedings of the AFIPS National Computer Conference, Vol. 45, AFIPS Press, Montvale, NJ, 1976, pp. 935-937.

[11] Grier, S.: "A Tool that Detects Plagiarism in Pascal Programs", SIGSCE Bulletin, Vol. 13, No. 1 (February 1981), pp. 15-20.

[12] Halstead, M.: "A Theoretical Relationship Between Mental Work and Machine Language Programming", Computer Science Department Internal Report CSD-TR 67, Purdue University, (February 1972).

[13] Halstead, M.: "Natural Laws Controlling Algorithm Structure", SIGPLAN Notices, Vol. 7, No. 2, (February 1972).

[14] Halstead, M.: "An Experimental Determination of the "Purity" of a Trivial Algorithm", ACM-SIGME: Performance Evaluation Review, Vol. 2, No. 1, (March 1973), pp. 10-15.

[15] Halstead, M.: "Software Physics: Basic Principles", RJ 1582, IBM, Yorktown Heights, N.Y., (1975).

[16] Halstead, M.: "Using the Methodology of Natural Science to Understand Software", Computer Science Department Internal Report CSD-TR 190, Purdue University, (1976).

[17] Halstead, M.: Elements of Software Science, Elsevier/North Holland, New York, (1977).

[18] Hanushek, Eric A. and John E. Jackson: Statistical Methods for Social Scientists, Academic Press, New York, (1977).

[19] Love, L. and A. Bowman, "An Independent Test of the Theory of Software Physics", SIGPLAN Notices, November, 1976, pp. 42-49.

[20] Ottenstein, K.: "A Program to Count Operators and Operands for ANSI-FORTRAN Modules", Computer Science Department Internal Report CSD-TR 196, Purdue University, (June 1976).

[21] Ottenstein, K.: "An Algorithmic Approach to the Detection and Prevention of Plagiarism", SIGSCE Bulletin, Vol. 8, No. 4, (December 1976), pp. 30-41.

[22] Shaw, M., A. Jones, P. Kneuven, J. McDermott, P. Miller and D. Notkin: "Cheating Policy in a Computer Science Department", SIGSCE Bulletin, Vol. 12, No. 2, (July 1980), pp. 72-76.

FEATURE	PROGRAM NUMBER 3	4	5	6	7	POOLED
Total Operators *	.98	.97	.99	.96	.94	.99
Assignment Statements	.97	.96	.97	.84	.72	.98
Total Operands *	.99	.96	.92	.76	.97	.95
Real Variables	.75	.81	.96	.68	.50	.91
Assignment Statements (Initialization)	.83	.73	.85	.69	.41	.90
Total Variables	.85	.83	.99	.89	.66	.87
Unique Operands *	.98	.92	.99	.55	.93	.85
Continuation Statements	.19	.65	.67	.01	.41	.82
Total Lines	.76	.73	.79	.78	.75	.78
Unique Operators *	.95	.88	.89	.46	.85	.75
Code Lines	.76	.90	.90	.98	.93	.75
Keywords	.84	.76	.91	.88	.85	.52
Declared Reals	.28	.07	.15	.58	.11	.21
Integer Variables	.28	.40	.86	.69	.55	.16
Declared Integers	.19	.36	.15	.10	.10	.14

* Halstead Feature

Table 1: Rotated First Factor Loadings by Feature and Assignment, Listed in Pooled Rank

FEATURE	ASSIGNMENT FACTOR RANK 3	4	5	6	7	MEAN RANK
Total Operators *	2.5	1	2	2	2	1.9
Total Operands *	1	2.5	6	7	1	3.5
Unique Operands *	2.5	4	2	10	3.5	4.4
Assignment Statements	4	2.5	4	5	8	4.7
Code Lines	9.5	5	8	1	3.5	5.4
Total Variables	6	7	2	3	9	5.4
Keywords	7	9	7	4	5.5	6.5
Unique Operators *	5	6	9	11	5.5	7.3
Real Variables	11	8	5	9	10	8.6
Total Lines	9.5	10.5	11	6	7	8.8
Assignment Statements (Initialization)	8	10.5	10	8	11.5	9.6
Continuation Statements	12	12	12	12	11.5	11.9

* Halstead Feature

$\overline{x} = 4.8$

Table 2: Factor Rank by Feature and Assignment

Assignment Number (Focus)	Halstead Method C = ⟨0,0,0,0⟩	Halstead Method C = ⟨1,1,1,1⟩	Alternative Method C = ⟨0,0,0,0⟩	Alternative Method C = ⟨1,1,1,1⟩
4 (Transfer of Control)	4 (75%)	8 (37.5%)	0 (--)	25 (8%)
5 (DO loops)	2 (50%)	9 (22.2%)	1 (100%)	7 (28.6%)
6 (Arrays)	4 (100%)	22 (18.2%)	5 (100%)	18 (61.1%)
7 (Subprograms)	2 (100%)	9 (53.6%)	2 (100%)	6 (100%)
Overall	12 (83.3%)	48 (29.2%)	8 (100%)	56 (39.3%)

Table 3: Detected Cases (% Correct) of Program Plagiarism by Method

BOTH	ALTERNATIVE ONLY	HALSTEAD ONLY
GOTO	DIMENSION	() -functions
DO	INTEGER	() - arrays
IF	REAL	'
THEN	CHARACTER	
WHILE	COMMON	
FOR	WRITE	
SUBROUTINE	RETURN	
FUNCTION	STOP	
LT	READ	
LE	FORMAT	
EQ	PRINT	
GT	CONTINUE	
GE	END	
NE		
AND		
OR		
NOT		

Table 4: Relationship Between Keywords and Total Operators in Two Metrics

```
      OVTIME = HOURS - 40
      IF (HOURS .GT. 40) GO TO 50
      OVTIME = 0
      GROSS = (HOURS * WAGE)
      GO TO 51
50    GROSS = (HOURS * WAGE) + (OVTIME * WAGE * 1.5)
51    OVPAY = (OVTIME * WAGE * 1.5)
```

Halstead Profile = ⟨24, 20, 8, 8⟩

Alternate Profile = ⟨7, 3, 8, 5⟩

```
      IF (HRS .LE. 40.0) GO TO 40
      OVTHRS = HRS - 40.0
      OVTPAY = OVTHRS * WGRATE * 1.5
      GRPAY = (40.0 * WGRATE) + OVTPAY
      GO TO 50
40    OVTHRS = 0.0
      OVTPAY = 0.0
      GRPAY = HRS * WGRATE
```

Halstead Profile = ⟨24, 20, 8, 8⟩

Alternate Profile = ⟨8, 3, 8, 6⟩

Appendix I

```
      CHARACTER * 15 AX
      CHARACTER * 8 DATE
      INTEGER EX, XND
      XTB = 0.0
      XTD = 0.0
      XTC = 0.0
      XTE = 0.0
      XTF = 0.0
      XTG = 0.0
      XTH = 0.0
      READ (5, 1) NOD
      READ (5, 2) DATE
 10   WRITE (6, 11) DATE
      DO 15 I = 1, NOD
 20   READ (5, 30) AX, BX, XND, DX, EX
      CX = 0.0
      DO 25 J = 1, 10
      READ (5, 31) XDH
      IF (XDH .LT. 0) GO TO 40
      CX = CX + XDH
 25   CONTINUE
 40   XAB = CX - 40.0
      IF (CX.LE. 40) XAB = 0
      XCC = XAB * DX * 1.5
      IF (CX .LE. 40) XCC = 0.0
      XAA = (CX * DX) + (XAB * DX * 1.5)
      FX = XAA * 0.05
      IF (XND .EQ. 0) GX = .15 * XAA
      IF (XND .EQ. 1) GX = .13 * XAA
      IF (XND .EQ. 2) GX = .11 * XAA
      IF (XND .EQ. 3) GX = .11 * XAA
      IF (XND .EQ. 4) GX = .10 * XAA
      TA = GX + FX + EX
      XBB = XAA * .05
      XTB = XTB + CX
      XTD = XTD + FX
      XTE = XTE + XAA
      XTF = XTF + XBB
      XTH = XTH + XAB
      XTC = XTC + TA
      XTG = XTG + XCC
 15   CONTINUE
 50   WRITE (6, 60) BX, AX, XND, DX, EX, FX, GX, TA, XAA, XBB
 70   WRITE (6, 80) XTB, XTD, XTC, XTE, XTF, XTG, XTH
      STOP
      END
```

Halstead Profile = ⟨106, 104, 11, 38⟩*

Alternate Profile = ⟨59, 40, 25, 29⟩*

*these profiles apply to entire program. Documentation and format statements have been deleted for clarity.

Appendix IIA

```
      CHARACTER * 15 NAME
      CHARACTER * 8 DATE
      INTEGER EMPL
      REAL HRWK, WART, MISC, PAY
      HSUM = 0.0
      TREX = 0.0
      TDED = 0.0
      TPAY = 0.0
      TNET = 0.0
      TOTH = 0.0
      TOTP = 0.0
      READ (5, 10) EMPL
      READ (5, 11) DATE
      WRITE (6, 100)
      WRITE (6, 20) DATE
      DO 1200 K = 1, EMPL
      READ (5, 12) NAME, ID, NUMB, WART, MISC
      HRWK = 0.0
         DO 1300 I = 1, 100
         READ (5, 13) HOURS
         IF (HOURS .LT. 0) GO TO 15
         HRWK = HRWK + HOURS
1300  CONTINUE
  15  OTHR = 0.0
      IF (HRWK .GT. 40.0) OTHR = HRWK - 40.0
      REG = HRWK
      IF (HRWK .GT. 40.0) REG = 40.0
      OPAY = (OTHR * WART * 1.5)
      GPAY = (REG * WART) + (OTHR * WART * 1.5)
      RETX = GPAY * .05
      IF (NUMB .EQ. 0) TAX = .15
      IF (NUMB .EQ. 1) TAX = .13
      IF (NUMB .EQ. 2) TAX = .11
      IF (NUMB .EQ. 3) TAX = .11
      IF (NUMB .GE. 4) TAX = .10
      GVTX = GPAY * TAX
      DEDS = MISC + RETX + GVTX
      NPAY = GPAY - DEDS
      WRITE (6, 200) ID, NAME, HRWK, WART, MISC, RETX, GVTX,
     XDEDS, GPAY, NPAY
      HSUM = HSUM + HRWK
      TREX = TREX + RETX
      TDED = TDED + DEDS
      TPAY = TPAY + GPAY
      TNET = TNET + NPAY
      TOTH = TOTH + OTHR
      TOTP = TOTP + OPAY
1200  CONTINUE
  63  WRITE (6, 300) HSUM
      WRITE (6, 400) TREX
      WRITE (6, 500) TDED
      WRITE (6, 600) TPAY
      WRITE (6, 700) TNET
      WRITE (6, 800) TOTH
      WRITE (6, 900) TOTP
1000  STOP
      END
```

Halstead Profile = ⟨106, 104, 12, 38⟩*

Alternate Profile = ⟨87, 55, 28, 31⟩*

*these profiles apply to entire program. Documentation and format statements have been deleted for clarity.

Appendix II-B

```
      REAL MONP, INTR, INT
      INTEGER PERIOD
      DIMENSION TABLE (9, 8), INTR (8), MONP (9)
      READ (5, 10) NUMTAB
      INT = .10
      DO 100 J = 2, 8
      INTR (J) = INT
      INT = INT + .01
100   CONTINUE
      PAY = 200
      DO 200 N = 1, 9
      MONP (N) = PAY
      PAY = PAY + 50
200   CONTINUE
      DO 500 K = 1, NUMTAB
      READ (5, 20) PERIOD
      DO 350 L = 1, 9
      DO 400 M = 2, 8
      TABLE (L, M) = (MONP (L) * PERIOD) / (1 +
     x  (INTR (M) / 12.0)) ** PERIOD
400   CONTINUE
500   CONTINUE
      WRITE (6, 30) PERIOD
      WRITE (6, 40)
      WRITE (6, 50) (INTR (M), M = 2, 8)
      DO 500 N = 1, 9
      WRITE (6, 60) MONP (N), (TABLE (N, I), I = 2, 8)
500   CONTINUE
      WRITE (6, 70)
600   CONTINUE
      STOP
      END
```

Halstead Profile = $\langle 40, 45, 7, 21 \rangle$*

Alternate Profile = $\langle 40, 31, 13, 9 \rangle$*

*These profiles apply to entire program. Documentation and format statements have been deleted for clarity.

Appendix III-A

```
      INTEGER NOT, NOE, LP, NOR, MP, NOC
      REAL PRIN, AIR (8), COUNT, ARRAY (9, 8)
      READ (5, 1) NOT
      COUNT = 0.10
      AIR (1) = 0.00
      DO 2 NOC = 2, 8
         AIR (NOC) = COUNT
         COUNT = COUNT + 0.01
 2    CONTINUE
      DO 9 NOE = 1, NOT
         READ (5, 3) LP
         MP = 200
         DO 5 NOR = 1, 9
            ARRAY (NOR, 1) = MP
            DO 4 NOC = 2, 8
               PRIN = (MP * LP) / (1 + AIR (NOC) *
     X               (1.0 / 12.0)) ** LP
               ARRAY (NOR, NOC) = PRIN
 4          CONTINUE
            MP = MP + 50
 5       CONTINUE
         WRITE (6, 6) LP, (AIR (NOC), NOC = 2, 8)
         WRITE (6, 7) ((ARRAY (M, N), N = 1, 8), M = 1, 9)
         WRITE (6, 8)
 9    CONTINUE
      STOP
      END
```

Halstead Profile = $\langle 40, 44, 8, 20 \rangle$*

Alternate Profile = $\langle 36, 25, 11, 12 \rangle$*

*These profiles apply to entire program. Documentation and format statements have been deleted for clarity.

Appendix III-B