

1976

An Algorithmic Approach to the Detection and Prevention of Plagiarism

Karl J. Ottenstein

Report Number:
76-200

Ottenstein, Karl J., "An Algorithmic Approach to the Detection and Prevention of Plagiarism" (1976). *Department of Computer Science Technical Reports*. Paper 142.
<https://docs.lib.purdue.edu/cstech/142>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

AN ALGORITHMIC APPROACH TO THE
DETECTION AND PREVENTION OF
PLAGIARISM

Karl J. Ottenstein
Computer Sciences Department
Purdue University
West Lafayette, Indiana 47907

CSD-TR 200
August 1976

AN ALGORITHMIC APPROACH TO
THE DETECTION AND PREVENTION OF PLAGIARISM

Karl J. Ottenstein

Computer Sciences Department
Purdue University
West Lafayette, Indiana 47907

The significant problem of detecting (nearly) identical student homework papers is non-trivial since a grader for a large class cannot remember all previously graded papers while examining the current one. This problem can be reduced by quantifying papers in such a way that equivalent ones are given equal values. Here we discuss one possible quantification which works well when applied to student computer programs.

The desired quantification is a function which maps the "homework space" into some value space. The ideal function, f , would impose a partitioning on the set of papers in the sense that if x and y are homework papers and the P_i are partitions of the homework space with $x \in P_i$ and $y \in P_j$, then $i=j$ iff $f(x)=f(y)$. If $f(x)=f(y)$ and x and y are unique, then one of x and y is a plagiarized version of the other. In other words, when all partitions have but one element, no cheating has occurred. This ideal function is unobtainable for several reasons: it is possible for identical work to be performed independently, the semantic equivalence of two items cannot always be shown deterministically, and there is a subjective area between plagiarism and paraphrasing.

Our task, then, is to find a good approximation to this function. The approximation should at least map all potentially equivalent homework papers into the same partition. It may not guarantee accuracy in that two papers being in the same partition will not imply that they are necessarily plagiarized. If P_1, P_2, \dots, P_n are the ideal partitions, our approximation should create Q_1, Q_2, \dots, Q_m where each Q_i is either some P_j or the union of several P_j 's. That is, the partitions are merely cruder.

The constant functions satisfy our requirements for an approximation since only one partition will be created; but, they do not simplify our initial problem since all elements must be individually inspected for cheating. A function which maps a homework paper into the integer representing its length in characters will invariably create numerous partitions, but they will not be the desired Q_i : the replacement of one token by a

synonym of a different length will place plagiarized assignments in separate partitions. A length function based on the number of tokens would eliminate this problem, but will still group together totally unrelated assignments simply because they have the same length. A function which takes into account some measure of the information content of a homework paper should give us more accurate partitions.

Any meaningful language can have its symbols classified into three sets:

- operators
- operands
- "syntactic sugar": symbols used only for readability

The information content of an element of a language, then, depends on the operators and operands, some function of which should lead to a good approximation to our ideal partitioning. This is simply a more formal description of the approach employed by [Bulut 1973].

In his study of student FORTRAN programs, Bulut counted the basic software science [Halstead 1972, 1977] parameters:

- η_1 - the number of unique operators
- η_2 - the number of unique operands
- N_1 - the total number of occurrences of operators
- N_2 - the total number of occurrences of operands

He noted that "the probability of using η_1 and η_2 symbols exactly N_1 and N_2 times in two different...[expressions] is very slim." Plagiarized copies were found by hand checking programs with identical η_1 , η_2 , N_1 , and N_2 values. Bulut observed that, as with the length function above, the results of this method are not affected by changes to operand names since such changes will not modify η_2 or N_2 .

A program to count these four parameters for FORTRAN modules was written [Ottenstein 1976] and used to confirm Bulut's work. Table 1 shows the partitioning imposed on 47 student programs from CS 210 at Purdue University by the "software science method". In the formalism developed here, we consider this method a mapping of programs into 4-tuples, $(\eta_1, \eta_2, N_1, N_2) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$, where \mathbb{N} denotes the set of natural numbers. Two partitions (A and B) have two programs in them; the rest have one. One program in partition B is a copy of the other, with slightly different comments and margining. The other pair is not as immediately detectable as being plagiarized because one author apparently changed all of the variable names and label numbers. Other programs with close correspondence of the parameters were

compared, but without positive results. Thus it seems that a good partitioning was obtained. Copies of the programs in partition A are included in Appendix A with the parameter counts.

The size of a program (in tokens) is given as column N in Table 1. Since $N=N_1+N_2$, the partitions created by the length function mentioned above are supersets of those created by the software science method. Here, the length function creates 10 partitions of size greater than one, while the software science method seems to have given us the ideal partitioning. So at least in this case, the additional information provided by η_1 and η_2 is well worth the small effort required to obtain it.

Bulut called the chances of two student programs having equal 4-tuples "slim". We can get a more quantitative probability estimate by observing that η_1 , η_2 , N_1 , and N_2 all appear to have somewhat normal distributions, in agreement with our intuition. (Appendix B gives the histograms for the four parameters.) In our particular sample, we have:

	<u>mean</u>	<u>median</u>	<u>mode</u>	<u>s.d.</u>	<u>min</u>	<u>max</u>
η_1	17.00	17	15	2.07	13	24
η_2	35.38	35	35	3.93	27	45
N_1	145.77	139	135	19.30	116	195
N_2	111.36	106	101	17.26	84	154

Assuming this normal distribution, there is clearly a greater likelihood of finding a pair of independently written programs with equal parameter values near the means as there is of finding such a pair with values on the tails. Thus, we can be more confident of a partition's accuracy as its individual parameter values approach the tails of their distribution curves.

Since the four parameters are not mutually independent, we use a multivariate normal density function, g , determined by the means vector, $m=(17.00, 35.38, 145.77, 111.36)$, and the covariance matrix $[C]$, $C_{ij}=E(X_i-m_i)(X_j-m_j)$ with $X=(\eta_1, \eta_2, N_1, N_2)$, to get a feel for the closeness of a 4-tuple to the means vector. The expression $g(X)/g(m)$ is 1 at $X=m$ and approaches 0 as we move away from the mean. Evaluated at the 4-tuple X for partitions A and B, this expression results in 0.45 and 0.018, respectively. This indicates that the programs in partition B are very probably plagiarized (the partition is accurate), while those in A are less probably so. Visual inspection of the programs is clearly warranted in any case, but one would be particularly suspicious of those in partition B. Since the accuracy of the partitions varies according to the location of the 4-tuples in the distribution space, it would seem advantageous to find a partitioning function whose range has a constant distribution. The existence of such a function is not

known at present, although one would expect that if such a function were found, it would not be particularly accurate. In general, meaningful measurements of human behaviour produce uneven distributions.

Many alterations made by students to copied programs will be transparent to this method. Cosmetic transformations such as the reordering time independent statements, recommenting, reformatting of text, and renaming variables and labels will have no effect at all on n_1 , n_2 , N_1 or N_2 . Most non-cosmetic alterations fall into one of six well-defined impurity classes¹, all of which are detectable by a slightly more sophisticated counter. Unfortunately, a student who cheated on only part of a program will not be detected.

Since the parameter counting routine was developed for other purposes, its \$300 or so development cost is not significant here: its running cost is about five cents (5¢) per 100 line student program on a CDC 6500. (This would be less were it not that the routine was written in ANSI-FORTRAN for portability and self-analysis.) Thus, this method of detecting plagiarism is both inexpensive and rapid. The preventive element mentioned in the title is simply the deterrent created by making it difficult to cheat successfully.

It seems that this method can not only be applied to programs in other computer languages, but to any assignment which requires the submission of written material. Of course, programs are the only practical item for measurement since they are already in machine-readable form, but software science has been applied with some success to English [Kuhl 1975, Halstead 1977] and one might hypothesize that similar results can be obtained there.

ACKNOWLEDGEMENTS

Special notes of thanks are due Dwight Andrews of Purdue University for collecting copies of his students' programs expressly for this study and to Professor Halstead, also at Purdue, for his encouragement and insights into software science.

¹The impurity classes are [Bulut 1974]:

- (1) self-cancelling operations
- (2) ambiguous usage of an operand
- (3) synonymous usages of operands
- (4) common subexpressions
- (5) unnecessary replacements
- (6) unfactored expressions

Module	n_1	n_2	N_1	N_2	N
35	16	34	116	84	200
14	16	27	121	94	215
8	15	29	125	93	218
17	15	33	125	93	218
1	19	33	131	93	224
26	15	34	132	96	228
22	19	32	135	93	226
31	16	32	132	98	230
27	18	32	135	95	230
41	18	32	135	95	230
45	16	35	133	99	232
44	17	36	131	101	232
47	20	33	134	98	232
11	18	32	129	105	234
2	17	37	136	101	237
16	15	36	137	101	238
39	15	35	138	101	239
36	17	28	139	103	242
32	18	36	139	103	242
40	19	37	141	101	242
20	17	35	139	105	244
28	15	37	139	107	246
5	16	35	138	108	246
12	16	35	135	111	246
21	16	36	135	111	246
13	18	35	142	104	246
24	15	36	135	113	248
43	18	34	146	106	252
6	15	33	143	111	254
18	15	42	144	112	256
9	16	35	139	117	256
4	21	35	150	106	256
38	16	29	141	117	258
10	19	35	149	109	258
33	16	40	143	121	264
46	18	37	153	120	273
3	18	41	163	121	284
19	13	39	151	127	288
30	20	40	162	126	288
23	15	36	158	133	291
29	19	31	180	120	300
15	14	41	170	142	312
34	17	39	179	142	321
7	24	34	182	143	325
42	15	40	195	147	342
25	19	45	193	154	347
37	19	45	193	154	347

(A)

Table 1:
47 student program parameter values as partitioned by the software science method (left) and the length function (right).

(B)

APPENDIX A

Source Listings of Programs in Partition B




```
1.      N=1
2.      IQ1=0
3.      IF1=0
4.      IP1=0
5.      IQ2=0
6.      IF2=0
7.      IP2=0
8.      C  HAVE REFERENCED THE COUNTERS
9.      READ 111, KQTS, KFTS, KPTS
10     111 FORMAT (I2, X, I2, X, I2)
11     READ 120, MAX
12     120 FORMAT (I2)
13     C  HAVE READ QUANTITIES ON HAND AND HEADER NUMBER
14     10 READ 100, NUM, ICODE, IQQTS, IQFTS, IQPTS
15     100 FORMAT (I1, X, I4, X, I2, X, I2, X, I2)
16     C  HAVE READ A DATA CARD. THE THREE SUCCEEDING IF STATEMENTS
17     C  CHECK ORDER QUANTITIES AGAINST QUANTITIES ON HAND.
18     C  'INSUFFICIENT QUANTITY' RECEIPT PRINTED IF APPLICABLE
19     IF (IQQTS LE KQTS) GO TO 20
20     PRINT 200, NUM, ICODE
21     PRINT 205
22     PRINT 300
23     GO TO 44
24     20 IF (IQFTS LE KFTS) GO TO 30
25     PRINT 200, NUM, ICODE
26     PRINT 205
27     PRINT 300
28     GO TO 44
29     30 IF (IQPTS LE KPTS) GO TO 40
30     PRINT 200, NUM, ICODE
31     PRINT 205
32     PRINT 300
33     GO TO 44
34     C  IF ORDER CAN BE FILLED, COSTS ARE COMPUTED
35     C  AND A RECEIPT PRINTED
36     40 KQTS=KQTS-IQQTS
37     KFTS=KFTS-IQFTS
38     KPTS=KPTS-IQPTS
39     QCOST=6.05*FLOAT(IQQTS)
40     FCOST=4.15*FLOAT(IQFTS)
41     PCOST=2.25*FLOAT(IQPTS)
42     TOT=QCOST+FCOST+PCOST
43     IF(N.EQ.1) GO TO 66
44     PRINT 200, NUM, ICODE
45     GO TO 77
46     66 PRINT 201, NUM, ICODE
47     77 PRINT 210
48     PRINT 220, IQQTS, QCOST
49     PRINT 230, IQFTS, FCOST
50     PRINT 240, IQPTS, PCOST
51     PRINT 250, TOT
52     PRINT 300
53     C  AFTER THE RECEIPT IS PRINTED, THE COSTS FOR EACH STORE ARE
54     C  UPDATED TO BE RECALLED AS A SUMMARY WHEN ALL CARDS ARE READ.
55     C  SUMMARY VARIABLES HAVE APPROPRIATE SUFFIXES.
56     C  1 FOR STORE NUMBER 1 AND 2 FOR STORE NUMBER 2.
57     IF(NUM.EQ.1) GO TO 33
58     IQ2=IQ2+IQQTS
```

```

59.      IF2=IF2+I0FTS
60.      IP2=IP2+I0PTS
61.      QC052=6.05*FL0AT(I02)
62.      FC052=4.15*FL0AT(IF2)
63.      PC052=2.25*FL0AT(IP2)
64.      GT0T2=QC052+FC052+PC052
65.      GO TO 44
66. 33 IQ1=IQ1+I0QTS
67.      IF1=IF1+I0FTS
68.      IP1=IP1+I0PTS
69.      QC051=6.05*FL0AT(IQ1)
70.      FC051=4.15*FL0AT(IF1)
71.      PC051=2.25*FL0AT(IP1)
72.      GT0T1=QC051+FC051+PC051
73. 44 N=N+1
74. C   THE NEXT STEP CHECKS THE CARD COUNT AGAINST THE HEADER
75.      IF(N.LE.MAX) GO TO 10
76.      PRINT 260
77.      PRINT 210
78.      PRINT 220, IQ1, QC051
79.      PRINT 230, IF1, FC051
80.      PRINT 240, IP1, PC051
81.      PRINT 270, GT0T1
82.      PRINT 300
83.      PRINT 280
84.      PRINT 210
85.      PRINT 220, IQ2, QC052
86.      PRINT 230, IF2, FC052
87.      PRINT 240, IP2, PC052
88.      PRINT 270, GT0T2
89. 200 FORMAT ('0',15X,'STORE ',I1,3X,'ORDER CODE ',I4)
90. 201 FORMAT ('1',15X,'STORE ',I1,3X,'ORDER CODE ',I4)
91. 205 FORMAT ('0', '*** ORDER NOT FILLED, '
92.      * ' INSUFFICIENT STOCK ON HAND ***')
93. 210 FORMAT ('0',14X,'ITEM',9X,'PRICE',5X,'COST')
94. 220 FORMAT ('0',12X,I2,' QUART(S)    $6.05  $',F6.2)
95. 230 FORMAT (' ',12X,I2,' FIFTH(S)   $4.15  $',F6.2)
96. 240 FORMAT (' ',12X,I2,' PINT(S)    $2.25  $',F6.2)
97. 250 FORMAT ('0',27X,'TOTAL $',F7.2)
98. 260 FORMAT ('1',17X,'STORE 1  TOTAL BILL')
99. 270 FORMAT ('0',21X,'GRAND TOTAL $',F7.2)
100. 280 FORMAT ('0',17X,'STORE 2  TOTAL BILL')
101. 300 FORMAT (' ', )
102.      STOP
103.      END

```

STATISTICS FOR THIS MODULE:

```

=====
OPERATOR  FREQUENCY
-----
E. O. S.      40
() OR DO      9
IF            6
*            9
+           13
-            3
=           29
.LE.         4
.EQ.         2
GOTO 20       1
GOTO 44       4
GOTO 30       1
GOTO 40       1
GOTO 66       1
GOTO 77       1
GOTO 33       1
GOTO 10       1
FLOAT        9
=====
    
```

ETA1= 18
N1= 135

```

=====
OPERAND    FREQUENCY
-----
TPT       1
N         5
KFTS      3
QC05T     2
QC051     2
QC052     2
PC05T     2
PC051     2
PC052     2
IQ0TS     5
NUM       1
GT0T1     1
GT0T2     1
IQPTS     5
IQFTS     5
    
```

```

-----
IP1        4
IP2        4
FC05T     2
FC051     2
FC052     2
IF1        4
IF2        4
KCTS      3
IQ1        4
IQ2        4
MAX        1
KPTS      3
1.00E+00   4
0          6
6.05E+00   3
4.15E+00   3
2.25E+00   3
-----
    
```

ETA2= 32
N2= 95

```

1. C          PROGRAM 2 CS 210
2. C
3. C    FOLLOWING CALCULATIONS ARE FOR D. T. W. D. PERTAINING TO WEEKLY SALES
4.    N=1
5.    IQTS=0
6.    IFIF=0
7.    IPTS=0
8.    IQT=0
9.    IFI=0
10.   IPT=0
11.   READ333, LQTS, LFTS, LPTS
12.   333 FORMAT(I2, X, I2, X, I2)
13.   READ330, NMAX
14.   330 FORMAT(I2)
15.   10 READ335, NUMST, IORC00, IQTB, IFIB, IPTB
16.   335 FORMAT(I1, X, I4, X, I2, X, I2, X, I2)
17. C    THIS DETERMINES WHETHER OR NOT THE ORDER CAN BE FILLED.
18.   IF(IQTB. LE. LQTS)GOTO11
19. C    IF THE ORDER CANNOT BE FILLED, THIS INFORMATION WILL BE PRINTED.
20.   PRINT100, NUMST, IORC00
21.   PRINT110
22.   GOTO55
23.   11 IF(IFIB. LE. LFTS)GOTO12
24.   PRINT100, NUMST, IORC00
25.   PRINT110
26.   GOTO55
27.   12 IF(IPTB. LE. LPTS)GOTO20
28.   PRINT100, NUMST, IORC00
29.   PRINT110
30.   GOTO55
31.   20 LQTS=LQTS-IQTB
32.   LFTS=LFTS-IFIB
33.   LPTS=LPTS-IPTB
34. C    FOLLOWING DETERMINES ALL COST INFORMATION IF ORDER CAN BE FILLED.
35.   QCOST=6. 05*FLOAT(IQTB)
36.   FCOST=4. 15*FLOAT(IFIB)
37.   PCOST=2. 25*FLOAT(IPTB)
38.   TOT=QCOST+FCOST+PCOST
39.   IF(N. EQ. 1)GOTO77
40.   PRINT100, NUMST, IORC00
41.   GOTO22
42.   77 PRINT101, NUMST, IORC00
43. C    THIS PRINTS OUT STORE ORDERS.
44.   22 PRINT111
45.   PRINT112, IQTB, QCOST
46.   PRINT113, IFIB, FCOST
47.   PRINT114, IPTB, PCOST
48.   PRINT115, TOT
49.   PRINT119
50.   IF(NUMST. EQ. 1)GOTO25
51.   IQT=IQT+IQTB
52.   IFI=IFI+IFIB
53.   IPT=IPT+IPTB
54.   QCOS=6. 05*FLOAT(IQT)
55.   FCOS=4. 15*FLOAT(IFI)
56.   PCOS=2. 25*FLOAT(IPT)
57.   GTOT=QCOS+FCOS+PCOS
58.   GOTO55

```

```
59.      25 IQTS=IQTS+IQTB
60.      IFIF=IFIF+IFIB
61.      IPTS=IPTS+IP7B
62.      QCOSQ=6.05*FLOAT(IQTS)
63.      FCOSQ=4.15*FLOAT(IFIF)
64.      PCOSQ=2.25*FLOAT(IPTS)
65.      GTOTQ=QCOSQ+FCOSQ+PCOSQ
66.      55 N=N+1
67.      IF(N.LE.NMAX)GOTO10
68. C     THIS PRINTS OUT THE TOTAL BILL.
69.      PRINT116
70.      PRINT111
71.      PRINT112, IQTS, QCOSQ
72.      PRINT113, IFIF, FCOSQ
73.      PRINT114, IPTS, PCOSQ
74.      PRINT117, GTOTQ
75.      PRINT119
76.      PRINT118
77.      PRINT111
78.      PRINT112, IQT, QCOS
79.      PRINT113, IFI, FCOS
80.      PRINT114, IPT, PCOS
81.      PRINT117, GTOT
82.      100 FORMAT('0',15X,'STORE',X,11,3X,'ORDER CODE',X,14)
83.      101 FORMAT('0',15X,'STORE',X,11,3X,'ORDER CODE',X,14)
84.      110 FORMAT('0','*** ORDER NOT FILLED. '
85.      1 'INSUFFICIENT STOCK ON HAND ***')
86.      111 FORMAT('0',14X,'ITEM',9X,'PRICE',6X,'COST')
87.      112 FORMAT('0',12X,I2,X,'QUART(S) $6.05 $',F6.2)
88.      113 FORMAT(' ',12X,I2,X,'FIFTH(S) $4.15 $',F6.2)
89.      114 FORMAT(' ',12X,I2,X,'PINT(S) $2.25 $',F6.2)
90.      115 FORMAT('0',27X,'TOTAL $',F7.2)
91.      116 FORMAT('1',15X,'STORE 1 TOTAL BILL')
92.      117 FORMAT('0',21X,'GRAND TOTAL $',F7.2)
93.      118 FORMAT('0',15X,'STORE 2 TOTAL BILL')
94.      119 FORMAT(' ')
95.      STOP
96.      END
```

STATISTICS FOR THIS MODULE:

```

=====
OPERATOR  FREQUENCY
-----
E. O. S          40
() OF DO         9
IF               6
*                9
+               13
-                3
=               29
LE              4
EQ              2
GOTO 11         1
GOTO 55         4
GOTO 12         1
GOTO 20         1
GOTO 77         1
GOTO 22         1
GOTO 25         1
GOTO 10         1
FLOAT          9
    
```

ETA1= 10
N1= 135

```

OPERAND  FREQUENCY
-----
TOT      1
N        5
OCOSQ    2
OCOS1    2
PCOSQ    2
PCOS1    2
GTOT     1
NMAX     1
TOTB     5
GTOTD    1
FCOS     2
IOTS     4
IPTB     5
OCOS     2
IFIB     5
    
```

```

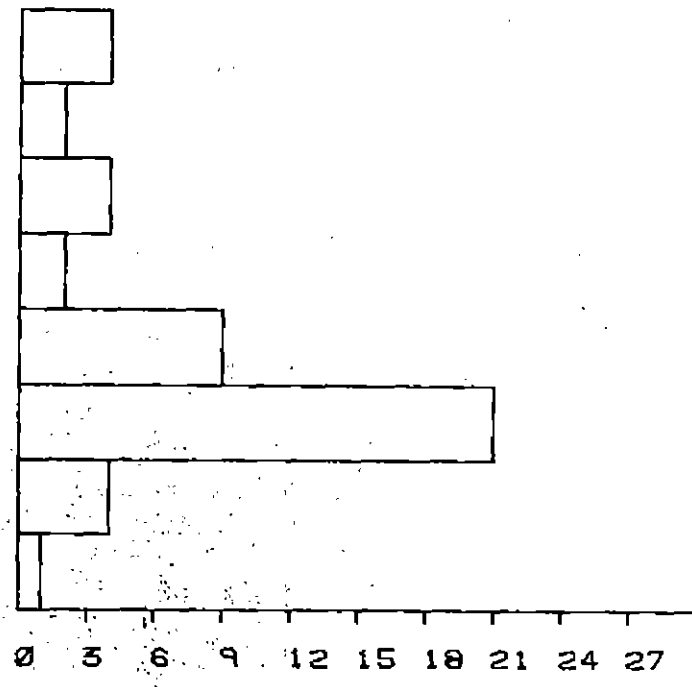
-----
IPTS     4
IFIB     4
IPT      4
LQTS     3
FCOSQ    2
FCOST    2
NUMST    1
LPTS     3
FCOS     2
IFI      4
LFTS     3
IQT      4
1.00E+00 4
0         6
6.05E+00 2
4.15E+00 2
2.25E+00 3
    
```

ETA2= 32
N2= 95

APPENDIX B

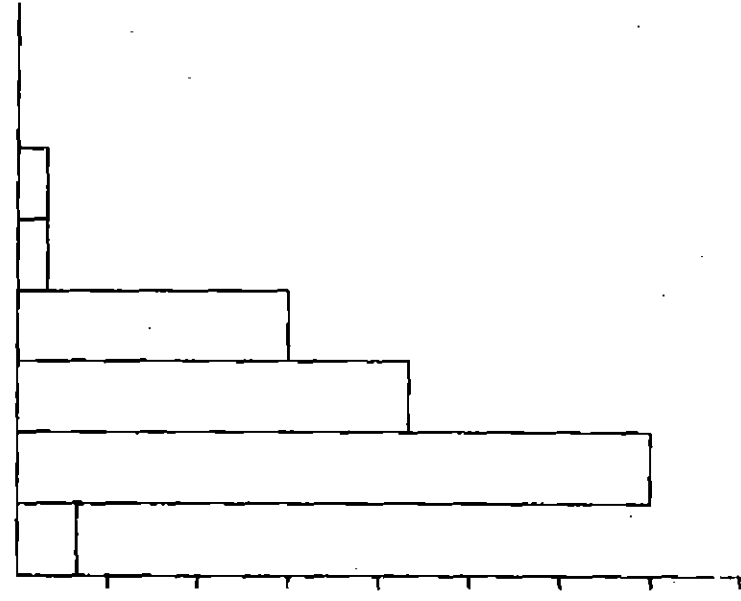
Histograms for μ_1 , μ_2 , N_1 , and N_2
for the Observed Sample

180.--	190.	4
170.--	180.	2
160.--	170.	4
150.--	160.	2
140.--	150.	9
130.--	140.	21
120.--	130.	4
110.--	120.	1



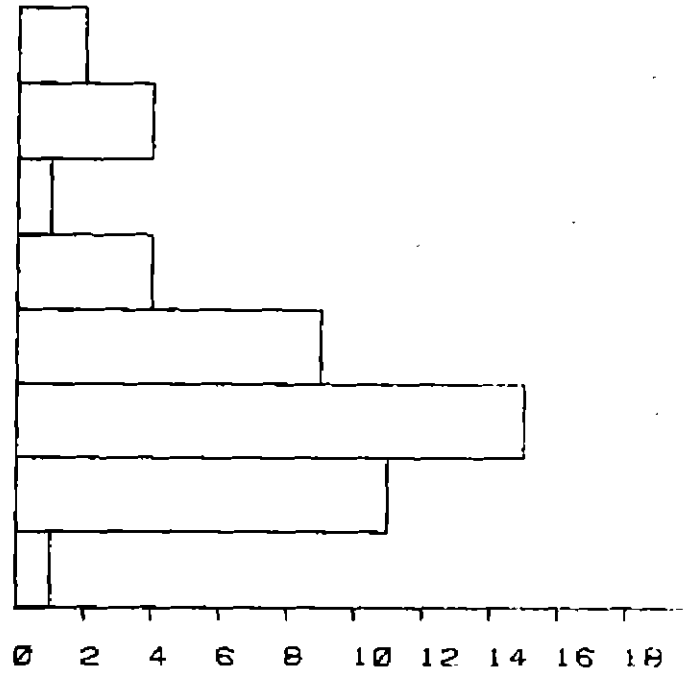
DISTRIBUTION OF N1 VALUES

26.--	28.	0
24.--	26.	0
22.--	24.	1
20.--	22.	1
18.--	20.	9
16.--	18.	13
14.--	16.	21
12.--	14.	2



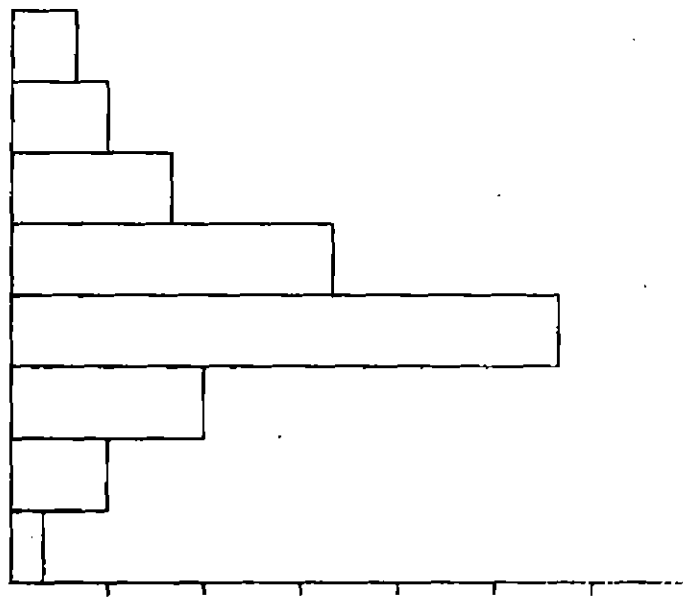
DISTRIBUTION OF ETA1 VALUES

150.--	160.	2
140.--	150.	4
130.--	140.	1
120.--	130.	4
110.--	120.	9
100.--	110.	15
90.--	100.	11
80.--	90.	1



DISTRIBUTION OF N2 VALUES

42.5--	45.	2
40.--	42.5	3
37.5--	40.	5
35.--	37.5	10
32.5--	35.	17
30.--	32.5	6
27.5--	30.	3
25.--	27.5	1



DISTRIBUTION OF ETA2 VALUES

References

- [Bulut 1973] Bulut, Necdet. Invariant properties of algorithms. PhD Thesis, Purdue University (August 1973) 118-119.
- [Bulut 1974] Bulut, Necdet and Halstead, Maurice H. Impurities found in algorithm implementations. CSD-TR 111, Purdue University (1974).
- [Halstead 1972] Halstead, M.H. Natural laws controlling algorithm structure? ACM SIGPLAN Notices 7, 2 (February 1972) 19-26.
- [Halstead 1977] Halstead, Maurice H. Elements of software science. Elsevier North Holland, New York. (1977) (in press).
- [Kulm 1975] Kulm, Gerald. Language level applied to the information content of technical prose. In Collective phenomena and the applications of physics to other fields of science (Prepared for delivery at a seminar, Moscow, USSR, 1-5 July 1974) Norman A. Chigier and Edward A. Stern, eds., Fayetteville, N.Y. Brain Research Publications (1975) 401-408.
- [Ottenstein 1976] Ottenstein, Karl J. A program to count operators and operands for ANSI-FORTRAN modules. CSD-TR 196, Purdue University (June 1976).