FOUNDATIONS AND EXPERIMENTS IN SOFTWARE SCIENCE

by

Nicholas Beser

23 Mar 82

General Electric Company

Space Division

Valley Forge Space Center

P.O. Box 8555

Philadelphia, Pa. 19101

ABSTRACT

A number of papers have appeared on the subject of software science; claiming the existence of laws relating the size of a program and the number of operands and operators used. The pre-eminent theory was developed by Halstead in 1972. The thesis work focuses on the examination of Halstead's theory; with an emphasis on his fundamental assumptions. In particular, the length estimator was analyzed to determine why it yields such a high variance; the theoretical foundations of software science have been extended to improve the applicability of the critical length estimator. This elaboration of the basic theory will result in guidelines for the creation of counting rules applicable to specific classes of programs, so that it is possible to determine both when and how software science can be applied in practice.

1.0 INTRODUCTION

Software metric requirements have increased from the simple measure of size and the execution speed of code to complex measures of program quality, and programmer productivity. With each new metric there is a growing need for standards, so that measuring improvement by comparison is possible. Software Science offers one form of software metric based on the software' product. This paper will discuss the problems found in the software science field, and will highlight some of the controversy found in this metric. The foundations of software science will be examined in an effort to highlight some possible causes of error in the metric. A mathematical framework will be setup so that effects of counting rule changes may be predicted. Some of the attributes of software, and software science, will be examined within the mathematical framework. An experiment will be performed to determine if the model suggested by the mathematical framework is supported empirically.

2.0 SOFTWARE SCIENCE

Software Science is a theory developed to measure properties of algorithms, and their implementation in languages [1,3,7-11,19,21]. By counting the tokens comprising a program: the operators and operands, software science attempts to measure quality and complexity of software; and the productivity of programmers. Operands are defined as, "the constants or variables that an implementation of code employs [21]." Operators are the "op-codes, delimiters, arithmetic symbols, punctuation, et cetera, which act upon the operands." In order to apply software science estimators, a set of rules for counting operators and operands must first be designed. A given piece of software is measured by applying the counting rules, and calculating various software metrics, which are based on the resulting parameter values. This methodology is applied to a carefully selected data base to calibrate the estimator equations forming an empirical basis for software science.

2.1 Anomalies Experienced With Software Science

Published research shows that software science measures may produce some interesting results with respect to the average. The variance of values over a set of benchmark programs is often too large to draw meaningful conclusions.[13,15] In particular, the length estimator, is significant only with respect to average values.

The length estimator[8] is an equation parameterized in terms of the number of unique operators and operands found in a program. The length estimator itself, has been suggested as a metric [3,4,8,11] for checking for well programmed software, as well as, for predicting effort, programming time, and expected number of errors. In the literature it was found that even for well programmed software, the length estimator constantly over estimates small programs and under estimates large programs.

While reviewing the published data, it was discovered that every experiment uses a counting rule, which is unique to the experiment. There has been some research at IBM, Purdue, University of Automata-Mexico and General Motors [1,3,5,12,18] into the effect of changing the counting rule. The counting rules seem to change with the focus of the investigation: from just measuring the algorithm (Halstead) to measuring the entire software package (Fitsos). Counting rules can change the magnitude and dynamic range of the measures. By counting more tokens, the volume can be increased and the program level can be decreased. To date, no one has been successful in devising counting rules that result in low variance. The differences in the counting rules used by various researchers makes comparing their empirical results a difficult task.

2.2 Problem Statement

2.2.1 Length Estimator -

The survey of the literature and past experiments raise many questions dealing with the length estimator, the counting rules sensitivity, and structured software. The readings and the experiments suggests that program size may be a critical factor when considering the performance of the length estimator. It would seem that small, medium, and large programs have different behavior. It would be of interest to see how the performance of the estimator changes with the counting rules. The results of the error analysis of the length estimator implies that there are really three different slopes to the curve. What is the meaning of the different slopes? Is it a defect in the metric? Is it an artifact of the counting rules? Is it a fundamental phenomenon of programming?

2.2.2 Counting Rule Sensitivity -

The question of counting rule sensitivity must be addressed. To what extent can counting rules be changed and effect the behavior of the curve? If counting rules are changed in a "reasonable" way will the distribution change? What is controlling the distribution? Is there a counting rule that would linearize the estimator? Is there a counting rule that would improve the variance of the estimator?

2.2.3 Structured Software -

In a recent paper on software science by Fitsos[1], he commented on the lack of growth in the number of unique operators in PL/1 code. Fitsos attributed this to the presence of structured software. He pointed out that the same measurements performed on assembly code did not observe a constant number of operator tokens. The question is, can a counting rule stress "quality" and "poor" programming? Would a counting rule take poor programming and make it look good by increasing or decreasing the variance? Do counting rules exist, which will be sensitive to only quality or poor programming?

2.3 Plan For Answering The Questions

Before attempting to answer these questions, the foundation for the metrics must be examined. Halstead and others [8-10,18,23] made critical assumptions about how software is created that bear directly on the variance of these measures. In particular, Halstead's assumptions do not hold for extremely large or extremely small programs. Language characteristics may also introduce new factors. It is very important to determine what types of software and languages fall into the categories defined by Halstead's model. This knowledge can be used to define when Halstead's measures are applicable. Smith[21] indicates in his survey that about 20 percent of his software falls into the area where Halstead's length metric is accurate.

A mathematical framework will be developed to model the measured length and length estimator equations. Elements of the equations, which could cause invariance or variance of the measures will be studied. The effect of token distributions, and size of a program will also be modeled. Finally, the combined effects of operator and operands will be included in the mathematical framework. The mathematical framework can be used to show the effect of counting rule changes on the length estimator.

A number of attributes connected with the length estimator will be examined. The percent variation of the length estimator with respect to the actual length will be viewed within the mathematical framework. This will help predict the effect of token distributions, and the counting rules role in the distributions. A measurement fundamental to the length estimator is the log base. The log base will be examined within the mathematical framework to see if it is token distribution dependent. Finally, the role of counting rules in structured programming will be examined.

The attribute level study will suggest a number of models that must be confirmed with a comprehensive experiment. The experiment will confirm or deny empirically whether the models are correct.

3.0 FOUNDATIONS OF SOFTWARE SCIENCE

In 1972 Dr. Maurice Halstead, at Purdue University, observed several quantitative relationships in computer programs. Halstead had been involved in decompilation work, taking object code and disassembling the code back into assembly code. He observed that the operator and operand token distribution seem to follow an inverse distribution. Halstead recognized that the same inverse distribution was desribed by Zipf[22] for natural languages. The regularity of token distributions suggested to Halstead that software could be measured at each stage of the software cycle, from high level language to assembly code.

3.1 Length Estimator

The software science theory developed by Halstead stated that as a program which consists of n unique tokens and N total tokens grows in size -- additional unique tokens are added -- the total tokens will grow $n*\log_2(n)$. Halstead used an analytical procedure and a probability model of software generation to develop the length estimator. He later used a combinatoric model as a means of deriving the length estimator. His derivation while taking into account the existence of operators and operands assumes no interaction between them. Halstead favorably compared both his theoretical model for software generation, and his length estimator to empirical data. All of the software science estimators were derived from the relationship shown by the length estimator.

3.1.1 Derivation Of The Length Estimator -

The Halstead-Bayer[10] derivation used identical procedures to formulate an estimator for the operator and operand terms. Informally, Halstead's length estimator equation was formulated as follows: the number of unique operators (n_1) and the total number of operators (N_1) could not be less than 2 because the shortest program consists of an assignment operator followed by the program terminator operator, in which case the ratio, $N_1/n_1=1$. Halstead observed that if one or more of the functional operators were added, the ratio would be greater than one. Halstead's basic assumption was that the rate of increase of N_1/n_1 with n_1 varies inversely with n_1, then the integration will produce a logarithmic term. If the smallest possible programs that can be written are allowed, the condition that $N_1/n_1=1$ at $n_1=2$ requires that the base equal 2, and hence the length estimator for the expected number of operators N_1 is given by the equation:

$$N_1 = n_1 \log_2(n_1)$$

Halstead's reasoning with respect to the estimator for the total number of operands is basically similar to the above: the number of unique operands (n_2) and the total number of operands (N_2) will behave in a similar fashion to the operators, yielding a total number of operand estimator N_2:

$$N_2 = n_2 \log_2(n_2)$$

Since the length N of a program is just the sum of the total number of operators and operands, the total length estimator (N) is given by the equation:

$$N\text{-estimate} = n_1\log_{b1}(n_1) + n_2\log_{b2}(n_2)$$

The base of the logarithmic terms may be calculated empirically by using the total tokens and the number of unique tokens, for example:

$$\text{base-operator-term}=e^{((n_1/N_1)*\ln(n_1))}$$

n_1 - number of unique operators, and
N_1 - total number of operators

Halstead concluded that since the rate of increase of N/n with n varies inversely with n, then the integration will produce a logarithmic term. The base would be a constant of two due to the initial conditions of the ratio N/n.

3.1.2 Halstead's Combinatoric Model -

Halstead's combinatoric model[8,9] for the length estimator assumes that in addition to the N tokens, the upper limit of the combination of n_1 and n_2 unique tokens include all possible subsets of the ordered set of tokens comprising a particular program. This was equated to the number of ways of selecting n_1 tokens from a group of n_1 unique tokens, and selecting n_2 tokens from n_2 unique tokens. The combinatoric model is an expression of a software generation process. Halstead did not take into account aspects of software that would change the number of combinations. Operator-operand interaction and operand span would tend to change the relationship implied by the model.

Halstead's probability model for the software generation process was used to justify the logarithm form of the estimator. Since the probability model is a classical occupancy problem, it derives to the poisson distribution, and thus supports the logarithm form. The model was analyzed, and found not to support the form of the logarithm base as a constant of two. The log base curves have been plotted in three dimensions (see Graph 1). Since the operator curve was symmetric with the operand curve, only one graph is presented.

The assumption is made by the author that the logarithm base is not constant, and is a function of the number of unique operators and operands. A survey of software was performed using the Purdue counting program. The program produced a logarithm base as follows:

$n_1>n_2$	b_1 approx. 3-5	b_2 approx. 1-3;
$n_1=n_2$	b_1 approx. 2+	b_2 approx. 3-4; and
$n_2>n_1$	b_1 approx. 1.0+	b_2 approx. 4-99.

The importance of the log base relationship is that if the token curves follow a constant inverse distribution, then the log base should be constant. The behavior of the software generation model and the empirical data suggests the converse is true. For different size programs, the token curves follows a distribution different from inverse. The change in the token distribution over size would account for the length estimator behavior.

The previous work focused on possible sources of error due to flaws in the foundations of software science. Since all software science metrics are performed by measuring software following some counting rules, there is still the question of effects of changes in the counting rules. There are many different types of counting rules. Each define operators, operands, create new tokens, or eliminate tokens from counting. What is needed is a mathematical framework that can be used to judge the effects of changes in counting rules. The model should also take into consideration changes in token distributions.

3.2 Mathematical Framework Of Length Measurement And Length Estimator

A mathematical model of the equations is developed; in order to discuss the effect of changes in the definitions of operators and operands on the length estimator. The model will be approached from the point of view of what kind of change could be made either in the length calculation or the length estimator that would cause one or both of the equations to vary. The relationship between the two equations is of interest since an explanation for the percent variation is desired. The length estimator is given by the equation:

$$P = n_1 * \log_2(n_1) + n_2 * \log_2(n_2)$$

Halstead Length Estimator Equation

where n_1 is the number of unique operators and n_2 is the number of unique operands. The length of the program may be described by the following equation:

$$L = \sum_{i=1}^{n_1} O_i + \sum_{i=1}^{n_2} V_i$$

Length of a Program Calculation

where O_i is the frequency of the operator tokens and V_i is the frequency of the operands tokens. The tokens n_1 and n_2 have the same definition as in the length estimator equation. The notation was changed from prior software science papers in order to easily distinguish between the length estimator and the measured length.

Both equations have operator and operand terms. Each term must be examined separately. It is recognized that once the individual terms are modeled, both terms may be combined in a limited way. There are many factors that could cause one equation to change while the other remains constant. Without entering into the question of counting rules, there are two elements of the equation that can vary; the number of unique tokens and the frequency of the individual token.

3.2.1 Operator Term Of The Equations -

The first term that will be examined will be the operator term. If two elements of the equations are varying, a problem can occur when they vary in the same sign. The problem becomes one of determining which variable is changing more. It is necessary to assume that the tokens are following a defined distribution. There are two major types of distributions considered in this model: Uniform, and Inverse. Study of software shows these two distributions are typical. This inclusion of token distributions allows size of a program, and whether a token is rare or frequent to be factored into the analysis.

The results of the math analysis is given in the table below.

		Uniform Distribution		
		Frequency of Tokens		
		-	0	+
Number	-	L>P	L>P	L>P
of	0	P>L	L=P	L>P
Unique Tokens	+	P>L	P>L	P>L

		Inverse Distribution		
		Frequency of Tokens		
		-	0	+
-	rare	L>P	L>P	L>P
-	freq	P>L		
0		P>L	L=P	L>P
+	rare	P>L	P>L	P>L
+	freq			L>P

At the points where the number of unique tokens and the frequency of unique tokens are changing in the same direction, the distribution of tokens will dictate how the two equations will relate. Size of the program did not effect the calculated relationship.

3.3 Attribute Level

There are three attributes connected with the length estimate that will be examined. The percent variation of the length estimator with respect to the calculated length not only gives a complete picture of the accuracy of the length estimator, but seems to differentiate between small, medium, and large programs. The log base is calculated from the unique number and total number of tokens. Experimental data indicates the operator log base shows the same size bias as the length estimator. The third attribute under study is the role counting rules play in scanning for structured software.

3.3.1 Percent Variation -

The percent variation of the length estimator has demonstrated a definite size bias. For small programs the estimator will over estimate, for medium size programs the estimator will be accurate, and for large programs the estimator will over estimate. The same behavior has been observed even when the definition of what constitutes a module is different. The IBM definition of module size is the entire assembly of subroutines and functions making up a product. The Purdue definition of a module is the unique subroutine or function.

There is the question of what is going on in the programs at the different size ranges. If a small program adds a previously used token, then the distribution of the tokens will dictate whether the length estimator will over estimate or under estimate the actual length. If the tokens distribution is uniform, then the length estimator will over estimate the actual length. For a uniform distribution, the ratio of total tokens to unique tokens is assumed to be a constant of one regardless of program size. This may occur only for small programs. If the token distribution is inverse, the question of whether a token is rare or frequent will determine the behavior of the estimator. If the token added is rare, the length estimator increase will be larger than the calculated length increase. If the token added is frequent, the calculated length could increase faster than the length estimate. When a program is large, the typical token added would be frequent. Since the metric is under estimating for large programs, it is also likely that the token curve follows an inverse distribution.

3.3.2 Length Estimator Log Base -

The log base was assumed to be a constant of two in Halstead's length estimator derivation. The constant value came about by considering the initial conditions for a small program. The minimum number of unique tokens must be two, and the total number of tokens must be two. The length estimator derivation implied that the rate of change of the ratio of total tokens to unique tokens will follow a inverse curve. The only way that the log base could be constant is if the ratio of total tokens to unique tokens follows a logarithmic curve. If the token distribution is uniform, a different behavior is observed. As new tokens are added to the set, the ratio of total tokens to unique tokens remains constant. The uniform distribution implies that the log base would rise as new tokens are added. The operator log base has been observed to change with the size of the program. This would imply that the operator token distributions also change as the program ranges in size. The operator log base for small programs has been observed to be greater than two. As the programs become larger, the log base has dropped below two. The operand log base, for small programs, has been observed to be greater than two. As the program size increases, the log base converges to approximately two. This would imply that the operand token distribution approaches inverse as the program size increases.

3.3.3 Relationship To Structured Software -

The two attributes that have been discussed deal only with the actual length metric. The differences between the predicted measurement and the actual measurement may be connected to changes in the token distribution. The implications of the token distributions to structured software is the third attribute that will be studied. In Fitsos's paper[1] the lack of growth of new operator tokens was attributed to the presence of structured PL/1 code. As a contrast, IBM assembly language was analyzed, and did not exhibit the same behavior. In addition, the length of the program was viewed as a function of the number of unique operands.

In order to evaluate whether the IBM counting rules stress structured software, it is first necessary to define what characteristics and tokens would be expected from structured and unstructured software. It is not the point of the paper to define structured software; however, there are some general characteristics that may be discussed. Structured software has the characteristic of being designed with a top down, hierarchical design approach, with block oriented code, and well defined communications to other routines.The structured code will have a definite communications path in the hierarchical design scheme, and would not use an excessive number of unique CALLs. In keeping with the block structured format, branching and GOTOs would be keep to a minimum.

In order to evaluate the conclusion that structured software has a constant set of operator tokens, it is necessary to examine the IBM counting rules. The counting rules are similar to the Purdue counting rules, with some major exceptions. CALLs to procedures are counted as the unique operator CALL. The module is defined as the entire collection of subroutines and functions. GOTO-labels are considered unique operators by both counting rules. A detailed example of the Purdue counting rule is found in the appendix.

It is clear from looking at the IBM counting rule, that if a branch to label was present, then the number of unique tokens would rise. If excessive CALLs is an example of unstructured code, and the IBM rule does not show the presence of the additional CALLS, then the conclusion that the PL/1 software is structured may not be correct. It is also possible that a counting rule may be devised that shows the same type of constant token behavior. By counting GOTO-labels as a unique operator GOTO and the unique operand LABEL, the operators may appear constant.

3.4 Summary Of Attributes

A number of conclusions may be drawn from the above analysis. The length estimator makes the assumption of constant form of a token distribution. The percent variation and log base attributes that have been observed would seem to suggest that the distributions are not constant. In addition, the conclusion that the unique operator tokens are constant for structured software are counting rule dependent. A experiment in counting rules has been performed to verify these conclusions. The experiment in counting rules will also demonstrate the effect of counting rule changes on token distributions.

4.0 EXPERIMENT

The purpose of the experiment will be to collect data that will either confirm or deny the theory that the operator and operand token distributions are the cause of the percent variation and log base error. The experiment will also demonstrate the relative effect on changing the counting rules by comparing the software science measurements of a data base with two very different counting rules. Two software analyzers are available which measure FORTRAN-IV source code: Purdue metric program and SAP/H program. The Purdue metric program is based on the paper by Karl Ottenstein[20] and has been expanded by Scott Woodfield, and was modified by the author to run on a VAX 11/780. The SAP/H program (Source Analysis Program/Halstead) was developed for Goddard Space Flight Center by CSC, and was converted to run on the VAX 11/780 by General Electric.

The two counting rules implemented in the Purdue metric program and the SAP/H program differ in several key areas: GOTOs, I/O, Data statements,and IF statements. A comparison of the two counting rules is given in the appendix. Both of the counting rule programs have been modified by the author to perform regression analysis[2] on each of the operator and operand distributions. The slope, Intercept, correlation coefficient, and F value of the regression is calculated. The SAP/H program also outputs a data base, which may be used to build a statistical picture of a module.

4.1 Data Selection

For the purposes of the experiment it is desired to select a data base that will guarantee that the results are statistically valid. The conclusions of the experiment should be stated with a high degree of confidence that the results are not subject to a reverse interpretation due to lack of significance of the data. Important questions while setting up the evaluating the data, will be: what results constitute an outlier; and what results are in an area where the data is not statistically significant.

The desired approach is to use published or production software. The data base will reflect software in a production and analysis environment, and will be from many sources. In addition to the General Electric software, the source to the International Mathematical and Statistical Libraries (IMSL) was available on the VAX 11/780. The IMSL source represents a product software package that is maintained by the leasing company. The source code largely has statistical and numerical analysis routines. The assumption is made that the IMSL library represents a single example of a large system that is under very tight configuration controls. The IMSL library is not claimed to be an example of totally structured software. However, the assumption is made that the IMSL library is of better quality then the random collection of simulators, assemblers, and analysis routines assembled as the General Electric data base.

4.2 Outputs From The Experiment

The two data bases have been evaluated by the Purdue and SAP/H counting programs. The results of the counting programs are presented in a number of graphs found in the appendix.

Length Estimate vs Length (Measured),
Percent Error of the Length Estimator vs Length (Measured),
Log Base of Operator Term vs Length (Measured),
Log Base of Operand Term vs Length (Measured),
Intercept of Operator Token Distribution,
Slope of Operator Token Distribution,
Intercept of Operand Token Distribution,
Slope of Operand Token Distribution,
Program Vocabulary Ranked by Program Length, and
Histogram of Operators and Operands

5.0 CONCLUSIONS

The data from the experiment is sufficient to draw some conclusions about the behavior of the length estimator and the role of counting rules in software science. The first group of questions that were asked, pertain to the size dependence of the length estimator and the log bases. The token distributions are suggested as the answer to the size dependence questions. The counting rules effect the type of distribution by either creating or eliminating rare and frequent tokens.

5.1 Percent Variation Of Length Estimator

In small programs, the length estimator over estimates the measured length. The mathematical framework suggests if the token distribution is closer to uniform, the length estimator will over estimate. This was supported by the experiment.

In medium sized programs, the length estimator is accurate. The length estimator derivation suggests if the token distributions are equal, then the length estimator will be an accurate metric. If the process generating the operand tokens is approached as a probability model, then from the literature, it would appear that a Markov model is appropriate. The mathematical framework, therefore, suggests the token distribution be an inverse function. This is supported empirically.

In a large program, the estimator under estimates the length. The choice of new operators becomes restricted to GOTO's, subroutine CALL's, and I/O statements. If the program size is still growing, the typical token encountered would be a frequent one. The mathematical framework suggests when the frequency of a token is increased, and the number of unique tokens is held constant, then the measured length will be greater than the estimated length. Since the length estimator is accurate for inverse distributions, the distribution may be approaching inverse squared. This is supported by the experiment.

The role that counting rules have in the token distribution, is the choice of creating frequent, less frequent, and rare tokens. The Purdue counting rule allows the existence of the rare tokens: GOTO-Label, I/O, and CALL-Label. This allows the set of operators to grow, thus keeping the slope of the distribution closer to inverse in the large case. The SAP/H counting rule only allows the rare token CALL-Label. The set of operators is restricted in its growth to frequent types and few rare types. The extreme number of frequent types force the token distribution slope much less than -1. It would appear that the SAP/H counting rule has a operator distribution slightly closer to the inverse squared, than the Purdue counting rule. Both counting rule operand distributions approach the inverse.

It may be concluded that the counting rule may alter the token distribution of large programs to the point where the operator token distributions are inverse. However, for cases where the program is small, the length estimator will probably always over estimate the length. The net result is: No change in counting rules will correct the anomalies observed in the length estimator.

The operand distribution curves show a great deal of variance compared to the operator distributions. When there is at least one distribution with a high variance, the estimator will have a high variance. The variance is possibly due to the open ended selection of unique operands. For very large programs, the operand distribution begins to look like a classic Zipf distribution. As in any statistical process, until the number of samples becomes large, the variance will be high. This implies the variance will never be reduced due to counting rule changes.

5.2 Log Base

The log base data was consistent with the prior analysis. When the token distribution is between uniform and inverse the log base is greater than two. When the token distribution has a steeper curve than inverse (approaching inverse squared) the log base will be below two. The log base length dependancy is the best indication that the token distribution is not a constant inverse as assumed by Halstead.

5.3 Structured Software

The question about screening software for structured and unstructured code was answered with a qualified yes. The counting rules could be setup to penalize the existence of tokens corresponding to unstructured code, if the definition of structured code is correct. A counting rule that counts each GOTO-Label, CALL-Label, and I/O as a new operators will see a growing set of tokens for unstructured code. It is possible to setup a counting rule that will produce a constant set of operators for structured and unstructured code.

5.4 Role Of Probability Models In Software Science

The distributions of operator and operand tokens make an interesting point about the software generation process and probability models. The software generation process is very complicated, and a model that attempts to predict every performance of the process will likely be inaccurate, due to the many variables, and unobserved facets of the problem. However, as in many language problems the process does seem to approach a consistent behavior as the program grows very large. This type of behavior can be modeled by a Markov process. If the counting rules are slanted, so that they favor a probability process rather than a psychological process, then for large programs, consistent performance of the metric might be observed. Counting rules based on cause and effect of a process must face the burden of proving the cause is correct. When a process that may be caused by a set of conditions is indistinguishable from a random process, then causality is difficult to prove. If a probability process is assumed, then a calculation of expected variance becomes possible. In addition, a framework which allows expansion for new metrics becomes possible. Unless a clear picture of the underlying process for software science is found, the only foundation that can be supported empirically is based on probability.

6.0 LENGTH ESTIMATOR PERFORMANCE

GE Software - Purdue Counting Rules - 870 Data Points Read

```
        REGRESSION ANALYSIS FOR LENGTH VS LENGTH ESTIMATOR

BO =     57.92957     +-    5.608710             80 % CONF LIM

B1 =    0.8141532     +-  1.0185177E-02          80 % CONF LIM

BO =     57.92957     +-    7.200943             90 % CONF LIM

B1 =    0.8141532     +-  1.3076604E-02          90 % CONF LIM

BO =     57.92957     +-    8.583219             95 % CONF LIM

B1 =    0.8141532     +-  1.5586756E-02          95 % CONF LIM

FVALUE =  10510.12    FOR    868.0000      DEGREES OF FREEDOM

XBAR =    308.4218      YBAR =     309.0322

RXY =   0.9610977    S =     106.8603
```

IMSL Data file - Purdue Counting Rules - 569 Data Points Read

```
        REGRESSION ANALYSIS FOR LENGTH VS LENGTH ESTIMATOR

BO =     79.36093     +-    20.56283             80 % CONF LIM

B1 =    0.6396490     +-  2.8854221E-02          80 % CONF LIM

BO =     79.36093     +-    26.40457             90 % CONF LIM

B1 =    0.6396490     +-  3.7051491E-02          90 % CONF LIM

BO =     79.36093     +-    31.47876             95 % CONF LIM

B1 =    0.6396490     +-  4.4171702E-02          95 % CONF LIM

FVALUE =  808.9830     FOR     567.0000      DEGREES OF FREEDOM

XBAR =    493.3111      YBAR =     394.9069

RXY =   0.7667642    S =     275.8971
```

<u>GE Software - SAP/H Counting Rules - 864 Data Points Read</u>

```
        REGRESSION ANALYSIS FOR LENGTH VS LENGTH ESTIMATOR

BO =     38.94392      +-   5.100722              80 % CONF LIM

B1 =   0.7810361       +-  1.1587721E-02          80 % CONF LIM

BO =     38.94392      +-   6.548766              90 % CONF LIM

B1 =   0.7810361       +-  1.4877360E-02          90 % CONF LIM

BO =     38.94392      +-   7.805895              95 % CONF LIM

B1 =   0.7810361       +-  1.7733281E-02          95 % CONF LIM

FVALUE =  7472.798    FOR    862.0000      DEGREES OF FREEDOM

XBAR =     250.2662      YBAR =     234.4109

RXY =   0.9468743     S =     96.16884
```

<u>IMSL Data file - SAP/H Counting Rules - 556 Data Points Read</u>

```
        REGRESSION ANALYSIS FOR LENGTH VS LENGTH ESTIMATOR

BO =     101.6805      +-   6.994767              80 % CONF LIM

B1 =   0.5058271       +-  9.1956351E-03          80 % CONF LIM

BO =     101.6805      +-   8.982014              90 % CONF LIM

B1 =   0.5058271       +-  1.1808158E-02          90 % CONF LIM

BO =     101.6805      +-   10.70823              95 % CONF LIM

B1 =   0.5058271       +-  1.4077513E-02          95 % CONF LIM

FVALUE =  4981.300     FOR    554.0000      DEGREES OF FREEDOM

XBAR =     506.0863      YBAR =     357.6727

RXY =   0.9486370     S =     95.96690
```

7.0 COUNTING RULE COMPARISONS

COUNTING RULE COMPARISONS

FORTRAN-IV element	PURDUE COUNTING RULE	SAP/H RULE
ACCEPT	Counted	Not Counted
BACKSPACE	Counted	Not Counted
CALL	Counted paired with routine name	Counted same as Purdue rule
DATA	Counted	Not Counted
DO	Counted	Counted, Paired with =,,
END	Not Counted	Not Counted
ENDFILE	Counted	Not Counted
GOTO LABEL	Counted As GOTO Lable	Counted as GOTO Label is operand
GOTO(),VAR	Counted	Counted
IF()STATEMENT	Counted, () seperate	Counted, grouped with ()
IF()LABEL,LABEL,LABEL	Counted, each lable is a seperate GOTO lable	Counted as IF() Labels not counted
PRINT	Counted	Not Counted
READ	Counted	Not Counted
RETURN	Counted	Not Counted
REWIND	Counted	Not Counted
STOP	Counted	Not Counted
TYPE	Counted	Not Counted
WRITE	Counted	Not Counted
Var=Expression	Evaluated and counted	Evaluated and counted
=	Counted	Counted, except from DO
Comma (,)	Counted, when in counted statement	Counted, when in counted statement
()	Counted, when in counted statement	Counted, from arithmetic express.
+ - * / **	Counted	Counted
Logical Operators	Counted	Counted
END OF STATEMENT	Counted	Counted
Function Calls	Counted as operators and operands	Counted as operators when used in arith statements, else as operands
´LITERAL STRINGS´	Counted as operands	Not Counted
Subscripts	Counted	Not Counted
Variables	Counted as operands	Counted as operands if in counted statements.
I/O Variables	Counted	Not Counted

8.0 BIBLIOGRAPHY

BIBLIOGRAPHY

[1]Christensen, K.,Fitsos, G. P., and Smith, C. P.,"A Perspective on Software Science", IBM Systems Journal, Vol 20, No.4, 1981.

[2]Draper, N., and Smith, H., Applied Regression Analysis, Wiley Interscience, 1966.

[3]Elshoff, J. L., Studies of Software Physics using PL/1 Computer Programs, General Motors Research Publication GMR-2444. June 1977.

[4]Elshoff, J. L., Measuring Commercial PL/1 Programs using Halstead's Criteria, General Motors Research Publication GMR-2012, Nov 1975.

[5]Fitsos, G. P., Software Science Counting Rules and Tuning Methodology, IBM Technical Report TR 03.075, September 1979.

[6]Fitsos, G. P., Vocabulary Effects in Software Science, IBM Technical Report TR 03.082, Jan 1980.

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

[8]Halstead, M. H., Elements of Software Science, American Elsevier, 1977.

[9]Halstead, M. H., Advances in Software Science, Advances in Computers, Vol. 18, 1980, Academic Press.

[10] Halstead, M.H., and Rudolf Bayer. Algorithm Dynamics, Proceedings of ACM Annual Conference, Atlanta, Aug. 1973. pp.126-135.

[11]Halstead, M. H., Gordon, R. D., Elshoff, J. L., On Software Physics and GM's PL/1 Programs, General Motors Research Publication GMR-2175, June 1976.

[12]Halstead, M. H., Zweben, S. H., The Frequency Distribution of Operators in PL/1 Programs, IEEE Trans on S/W Eng., Vol.SE-3, No.2, March 1979, p91-95.

[13]Hamer, P.G., Frewin, G. D., M. H. Halstead's Software Science - A Critical Examination, ITT Technical Report No. STL 1341, July 1981.

[14]Gordon, R. D.,Measuring Improvements In Program Clarity, IEEE Trans. On S/W Eng., Vol.SE-5, No.-2, March 1979.pp. 79-90.

[15]Johnston, D. B. and Lister, A. M.,(1979), "An Experiment in Computer Science", Proceedings of the Symposium on Language Design and Progamming Methodology, Sydney, 10-11 Sept. 1979; in Lecture Notes in Computer Science, Vol. 79, Springer-Verlag, 1980.

[16]Kavipurapu, K. M., Frailey, D. J., Quantification of Architectures Using Software Science, Computer Architecture News, Oct 1979, No. 10, p2-6.

[17]Kernighan, B. W., and Plauger, P. J., The Elements of Programing Style, McGraw Hill, N.Y., 1974, p. 108.

[18]Magidin, M., Viso, E., On the Experiments in Algorithm Dynamics, Technical Report, Dept of Math, Universidad Autonoma Metropolitana-Iztapalapa, Mexico, Vol. 1, No. 14., Oct 1976.

[19]Mohanty, S. N., Models and Measurements for Quality Assessment of Software, Computing Surveys, Vol.11, No.3, Sept 1979, pp. 251-275.

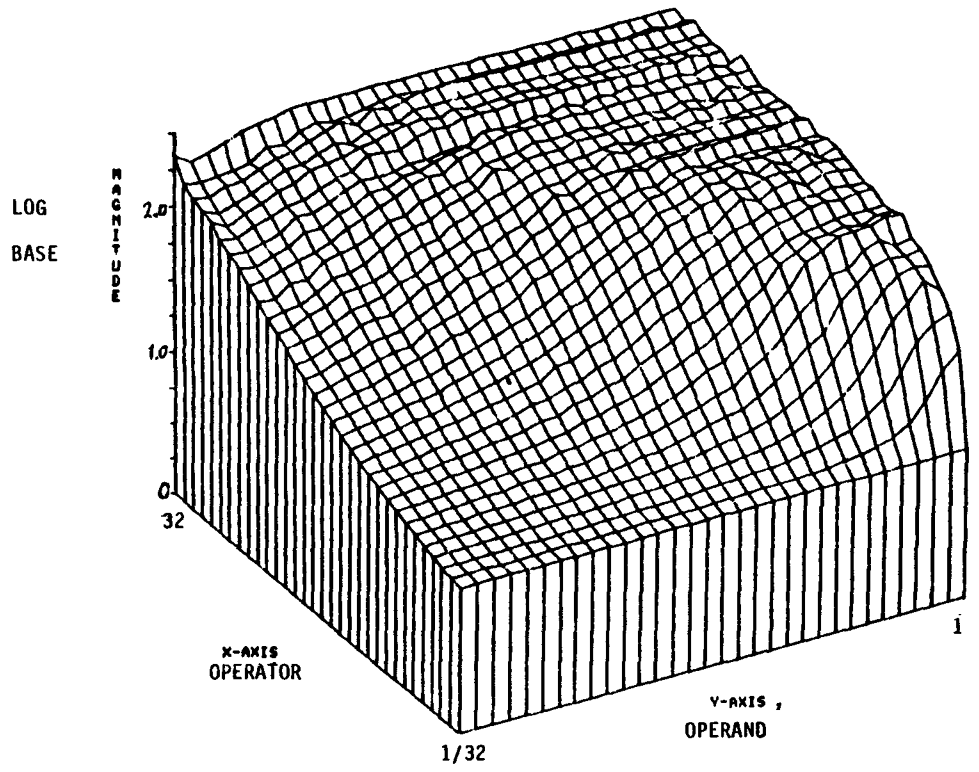
[20]Ottenstein, K. J., A Program to Count Operators and Operands for ANSI-Fortran Modules. Technical Report 196, Computer Science Department, Purdue University, June 1976.

[21]Smith, C. P.,A Software Science Analysis of IBM Programming Products, IBM Technical Report TR 03.081, Jan. 1980.

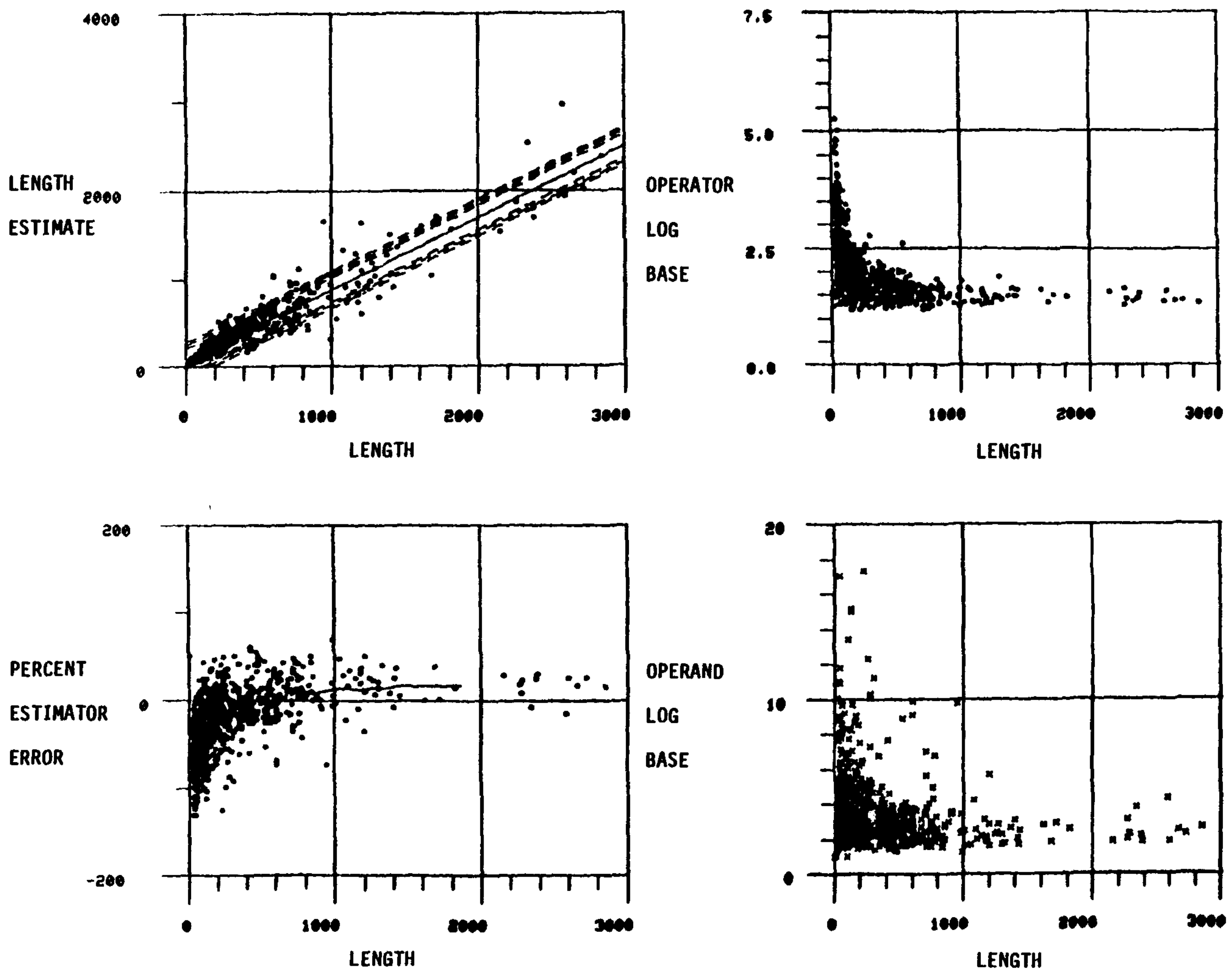
[22]Zipf, G. K. Human Behaviour and the Principle of Least Effort. Reading, MA: Addison-Wesley, 1949.

[23]Zweben, S. H., A Study of The Physical Structure of Algorithms, IEEE Trans on S/W Eng, Vol.SE-3, NO.3, May 1979, pp. 250-258.

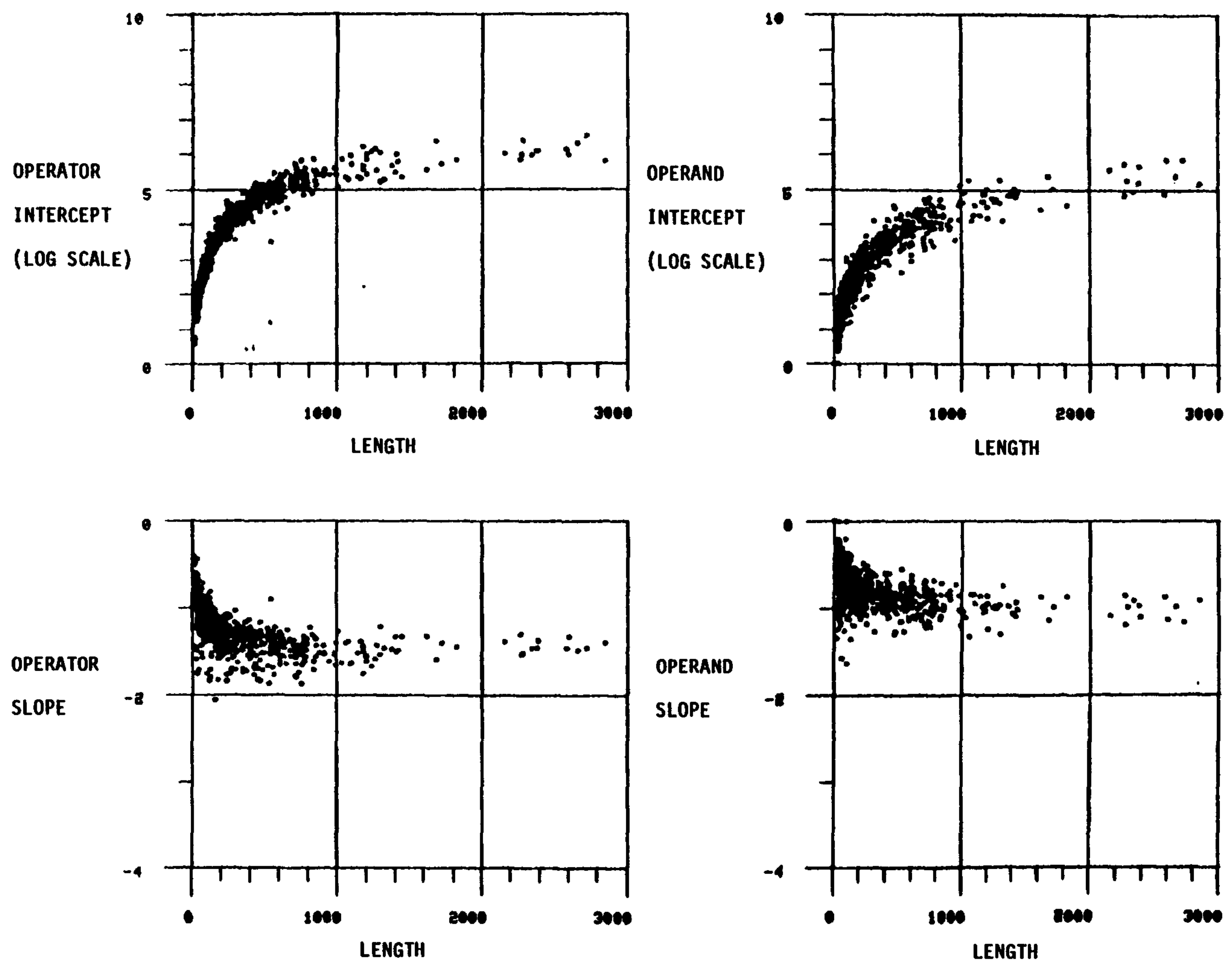
GRAPH 1 OPERATOR LOG BASE - FROM HALSTEAD/BAYER MODEL B



G E DATA BASE - PURDUE COUNTING RULES

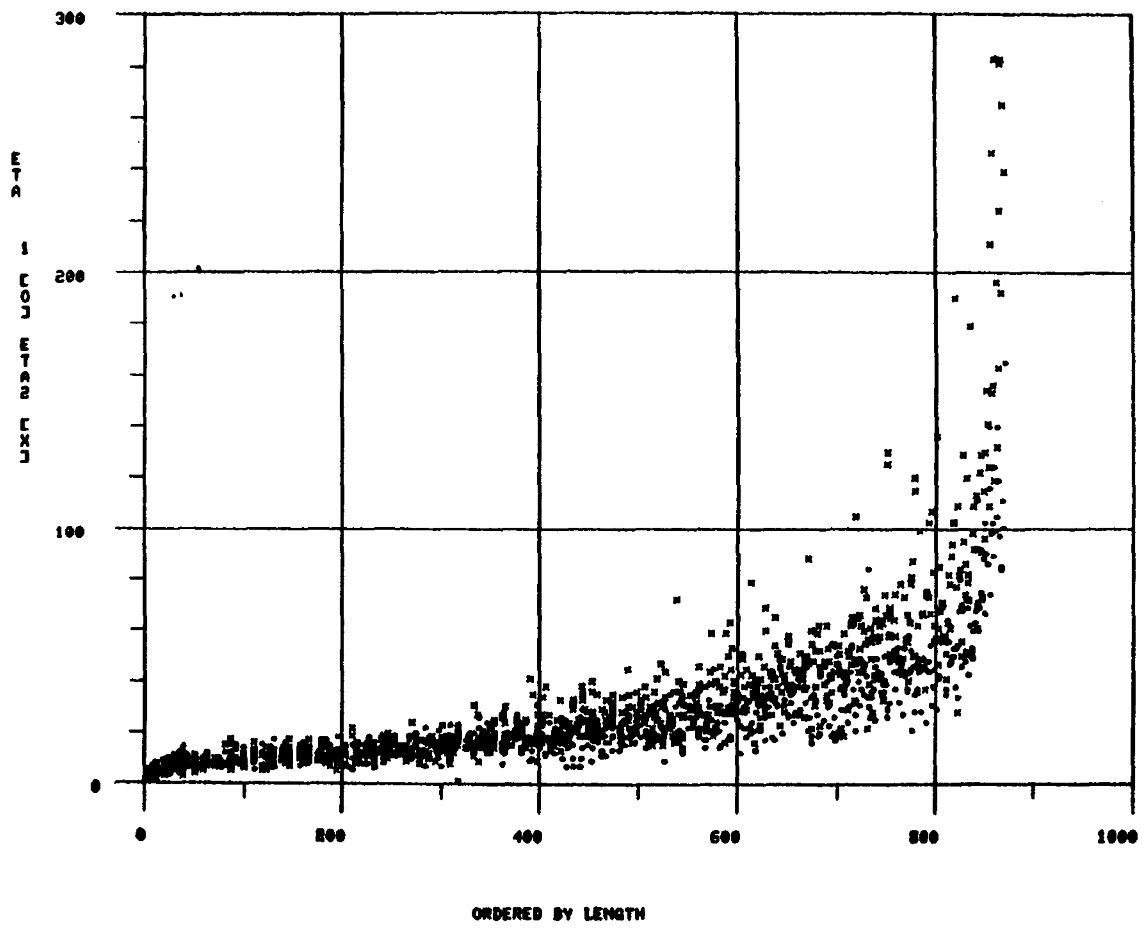


G E DATA BASE - PURDUE COUNTING RULES

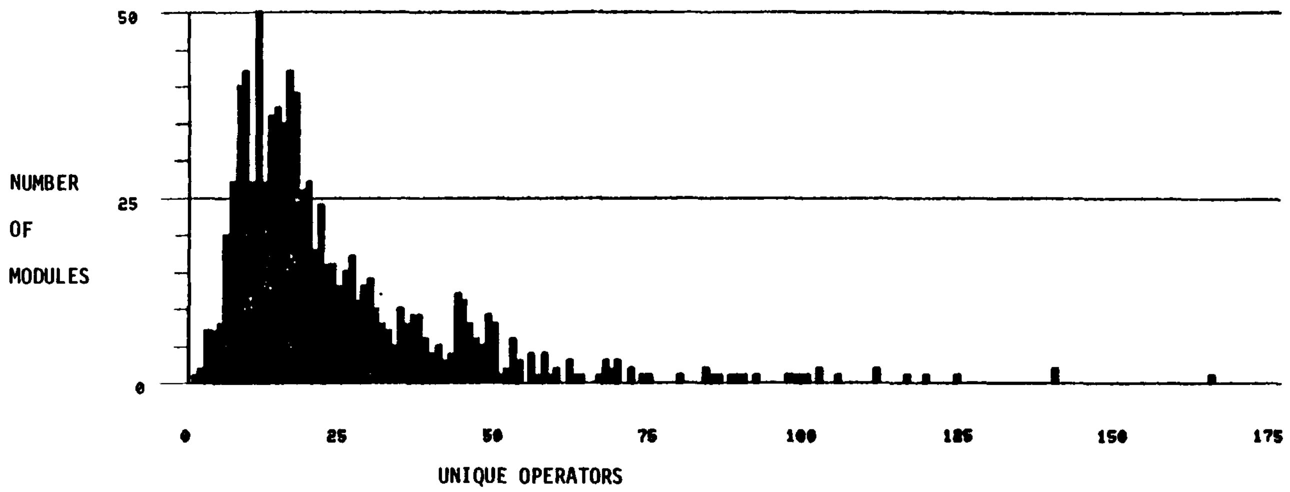


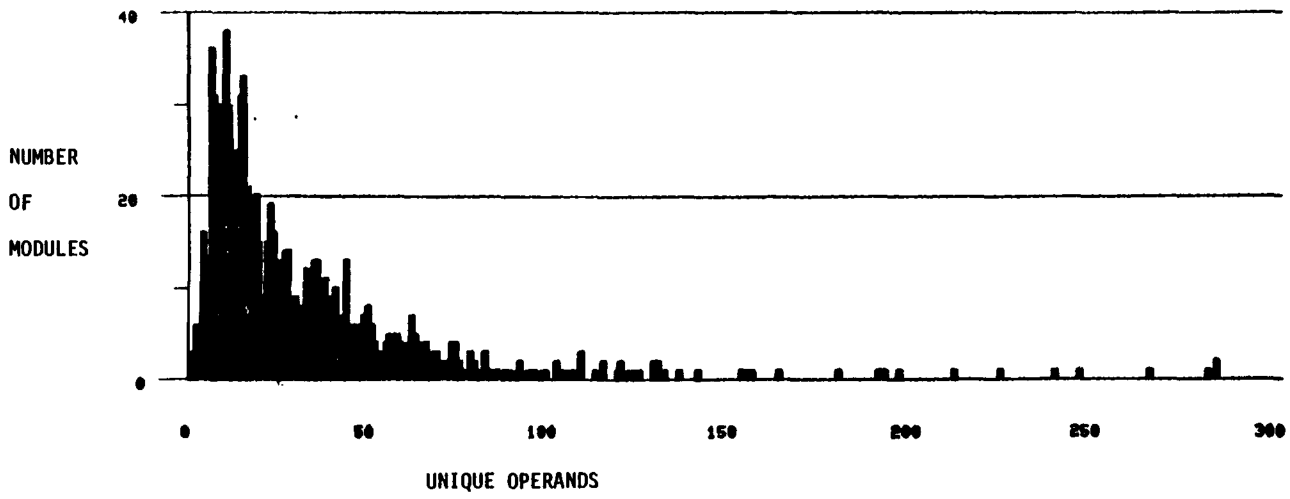
G E DATA BASE - PURDUE COUNTING RULES

ETA1 - ETA2 VS LENGTH



G E DATA BASE - PURDUE COUNTING RULES


NUMBER
OF
MODULES
UNIQUE OPERATORS


NUMBER
OF
MODULES
UNIQUE OPERANDS