THE VALIDITY OF A MARKOV MODEL OF THE BEHAVIOUR OF PROGRAMS

by

Edward Alfred Favre

Thesis submitted to the Graduate Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree
of MASTER OF SCIENCE
in
Computer Science and Applications

Approved:

T C Wesselkamper

Dr. Thomas C. Wesselkamper, Chairman

Dr. John A. N. Lee

R A Thompson

Dr. Richard A. Thompson

May, 1977
Blacksburg, Virginia

Contents

I. Introduction.

This thesis concerns one aspect of the behaviour of programs; the relative frequency of execution of source language statements in a program. Knowledge of how often certain parts of a program are executed relative to other parts may be used by the software engineer in a number of ways. Rauscher [1] has investigated the use of dynamic microprogramming in the optimization of program execution time. If the size of the control store is limited so that only a small portion of a program may be microprogrammed then the most frequently executed statements are best candidates for microcoding so as to provide improvement in the total execution time. Halstead's [2] work in software physics deals with the "purity" of algorithms (and programs) from the viewpoint of programming effort. In terms of software development management, the design of the most frequently executed routines and operations should be given the greatest amount of thought and research. Wesselkamper and Zoladz [3] suggest that the predicted behaviour may be compared with observed behaviour to detect and locate bugs in programs.

This report will present first a Markov Process model for program behaviour, recently used by Rauscher [1] and formalised by Wesselkamper and Zoladz [3]. A method

of using the model to accurately predict the behaviour of programs will then be developed and validated by comparing model predictions with runtime statistics for several sample programs.

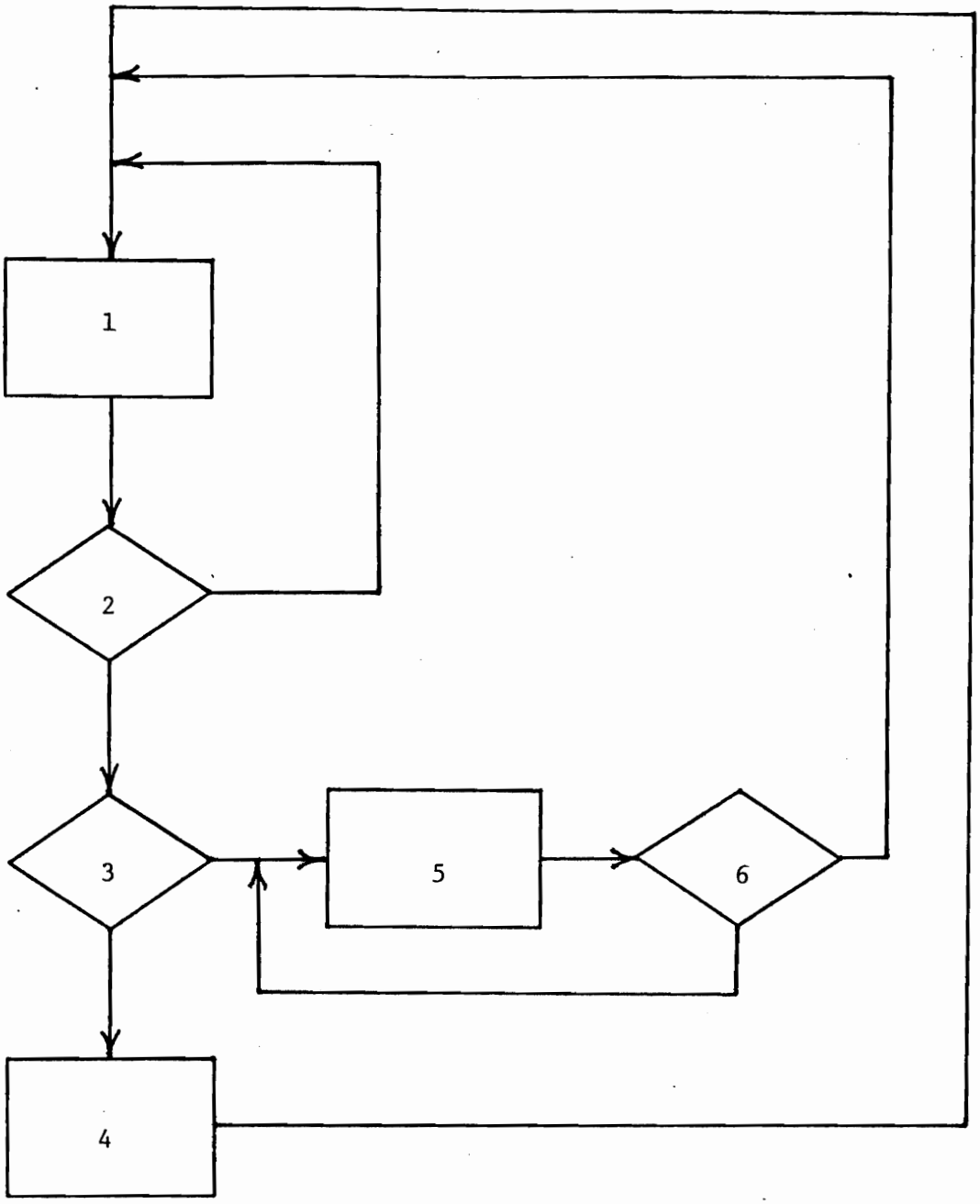
II. The Markov Model of a Program.

A. The Model.

The idea of modelling a program as a Markov process has been used by Rauscher [1]. The Markov Process used for this purpose is essentially a finite state machine with probabilities attached to the state transitions, but with no inputs or output functions. If the program structure is drawn as a flowgraph (composed of decision nodes, process nodes and execution paths), and the nodes are interpreted as states and the execution paths as state transitions, one can readily see the similarity of the Markov process to the flowgraph. As an example, take the flowgraph of Wirth's scanning routine (fig. 1) versus its representation as a Markov process (fig. 2). In figure 2, $p(ij)$ is the probability of moving to state j given that the process is in state i. This form of the Markov Model may be formally defined as

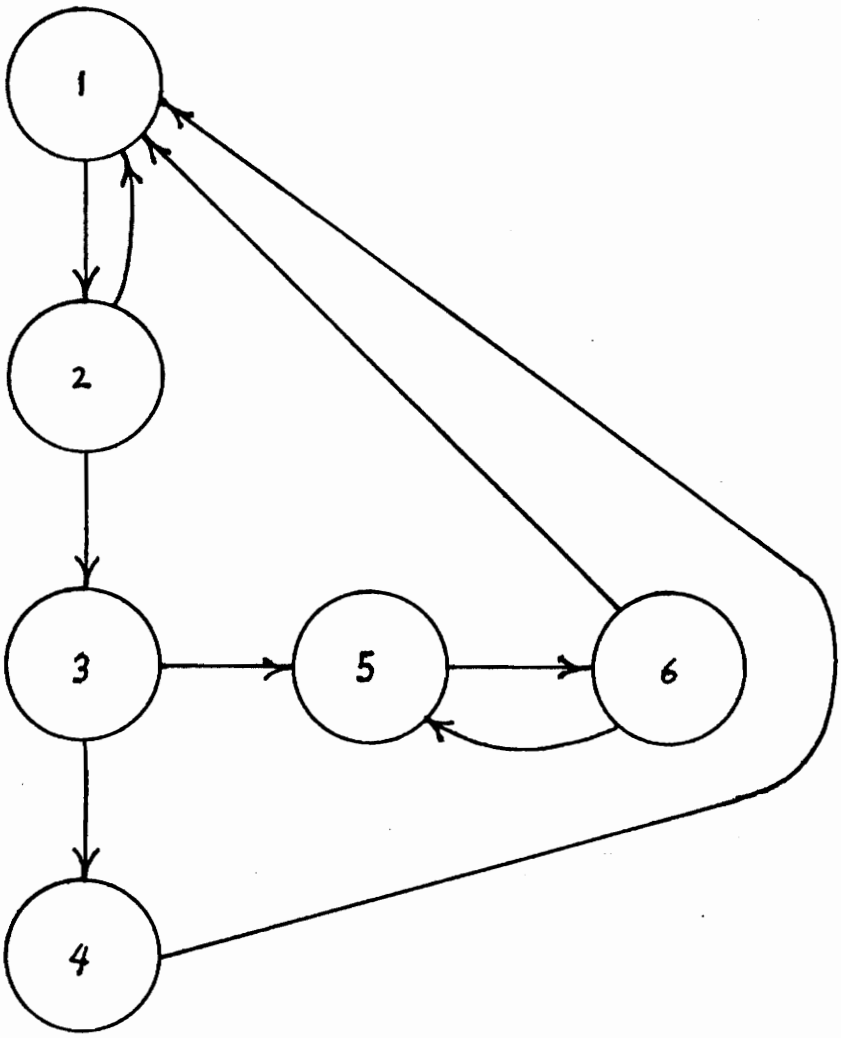
MM=(S,t) where S= the set of states

t= a function S x S -> p, $0 \leq p \leq 1$.



Flowgraph of Wirth's Scanning Routine

Figure 1.



Wirth's Scanning Routine as a Markov Process

Figure 2.

B. Restrictions on Program Structure.

There are some restrictions to the programs which may be modelled by the Markov Model. The restrictions are that the program be structured, global, and meet a relatively prime cycle condition. These were proven sufficient by Wesselkamper and Zoladz [3], and are explained in the next two sections.

B.1. Structured Programs.

Rauscher [1], and Wesselkamper and Zoladz [3] established the requirement that all programs which are to be modelled must be structured from basic programming constructs derived by Bohm and Jacopini [4], Dijkstra [5], and Martin [6] (fig. 3). This was done for reasons of good programming practice and discipline only, since the following section's restrictions are sufficient for a program to be modellable.

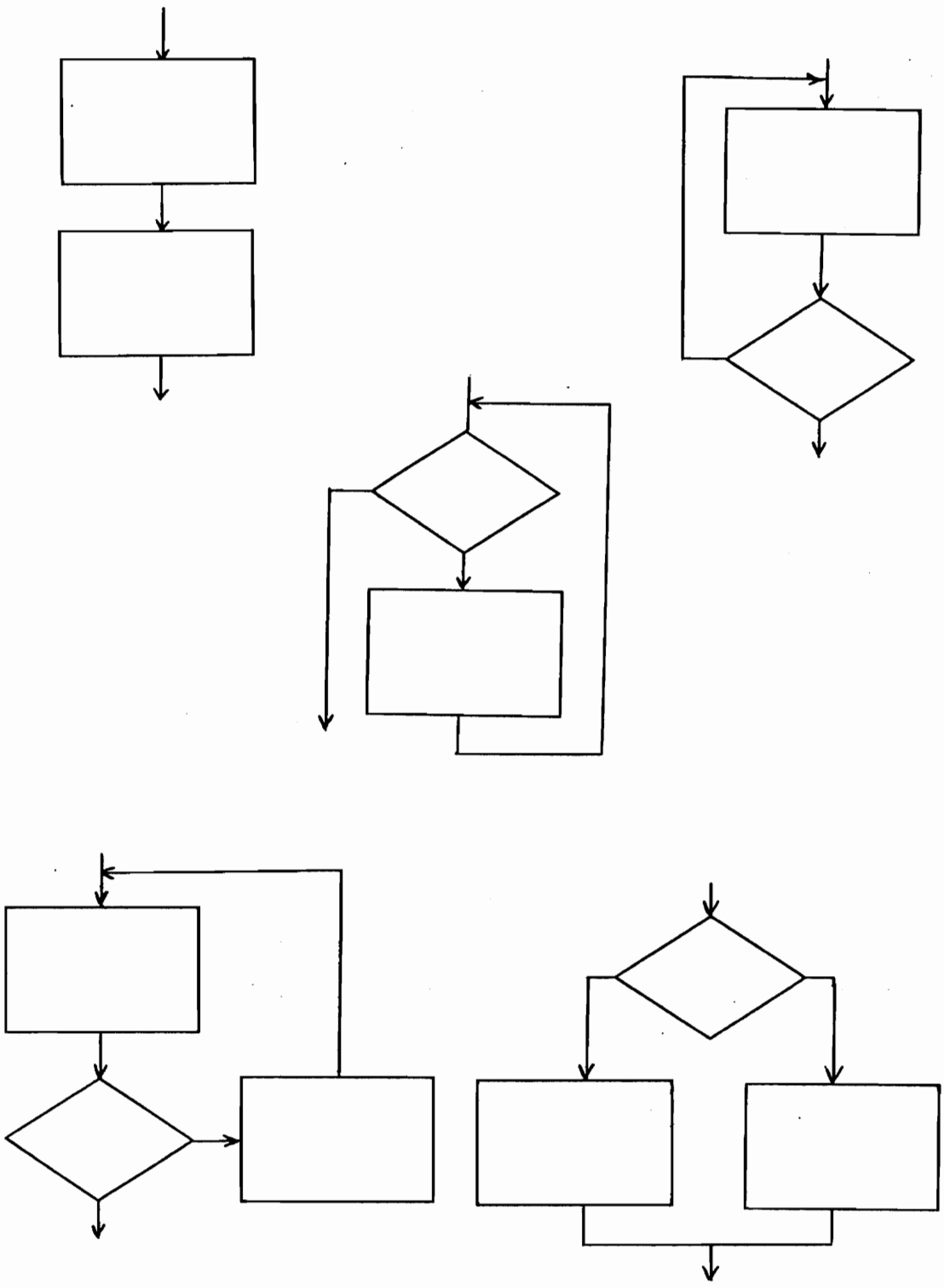


Figure 3. Structured Programming Constructs

B.2. Regular Flowgraphs.

The method used in the modelling procedure is based on the concept of the Finite Markov Chain. This method requires that the process be strongly ergodic. A flowgraph that is isomorphic (has the same structure) as a strongly ergodic Markov Process is called regular.

A finite state machine or Markov Process is ergodic if it is possible to move from any state to any state in the process in a finite number of state transitions. Ergodicity is guaranteed if the isomorphic flowgraph can be generated from the global goal (fig. 4) by successively applying structured construct productions (fig. 5) which also satisfies the condition that the program be structured. This is often called the top down method of program development. As an example, Wirth's scanning routine is generated by this procedure (fig. 6).

To insure that the process is strongly ergodic, Wesselkamper and Zoladz [3] introduce the relatively prime cycle (rpc) condition. The rpc condition requires that at least one state in the process have two paths from itself to itself (cycles) whose path lengths (number of transitions) are relatively prime (have 1 as the greatest common divisor). This property is illustrated in figure 7 where the second flowgraph is not regular because all

cycles for all states have lengths which are divisable by 2. The first flowgraph is regular because two cycles of the starred node have relatively prime lengths of 3 and 4.

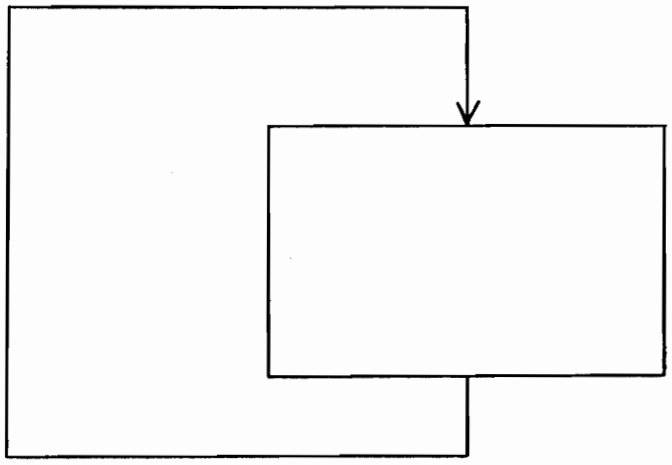


Figure 4. Global Goal

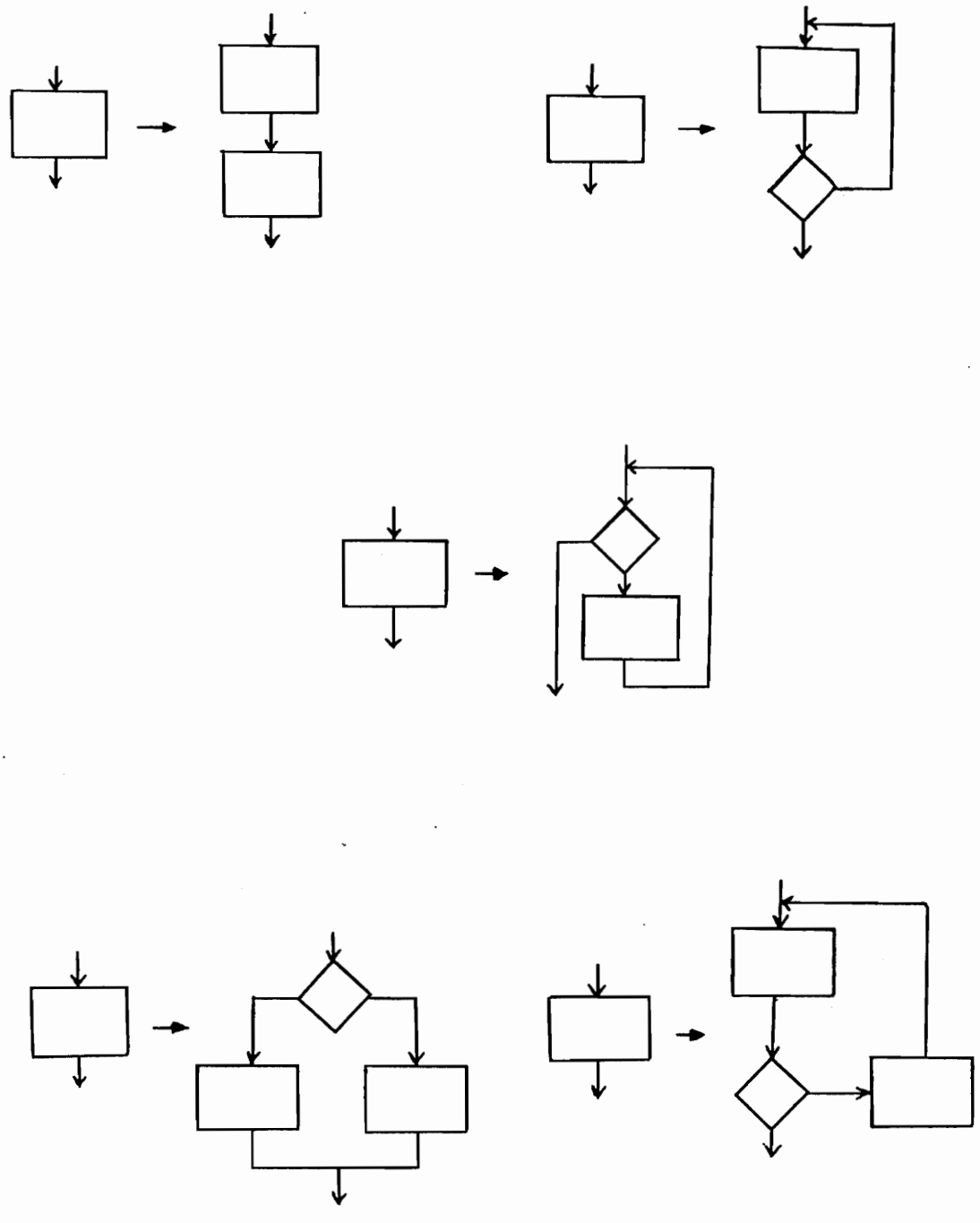
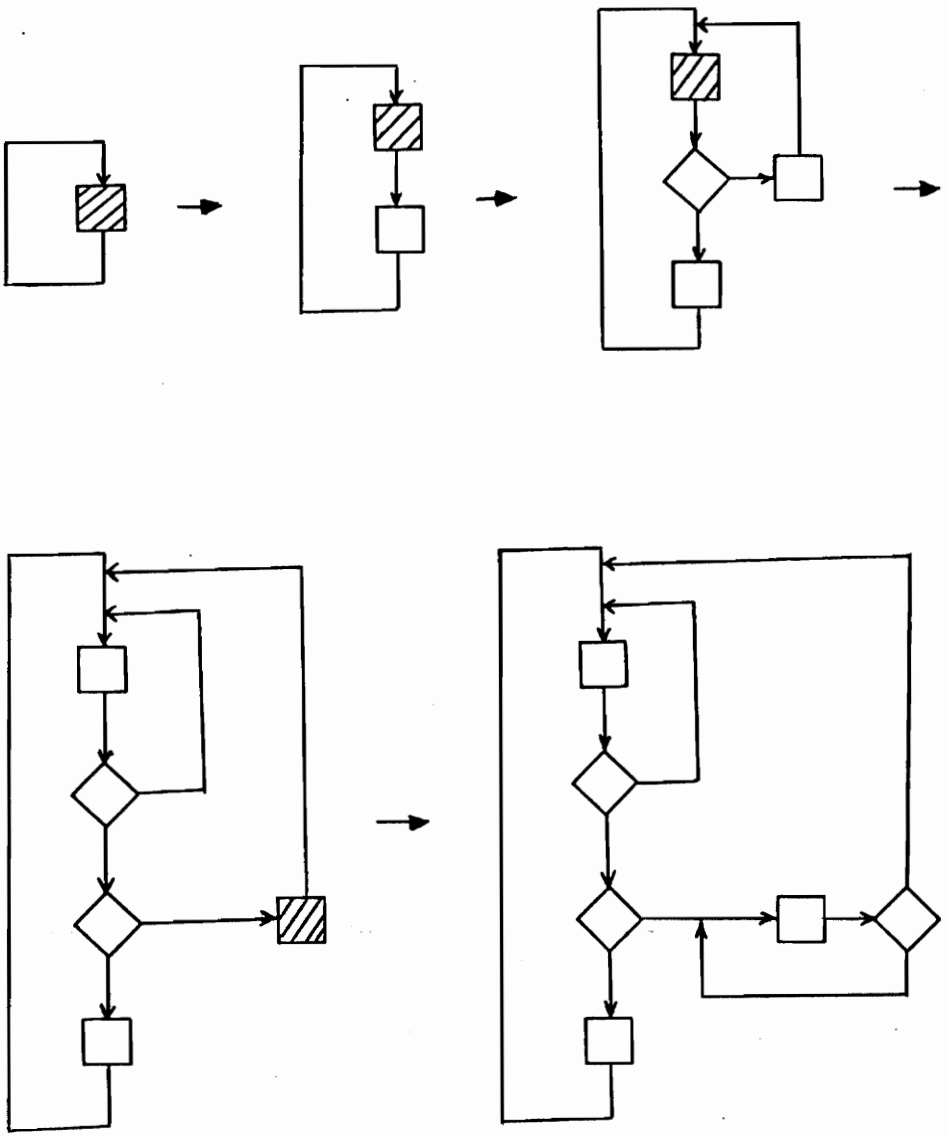
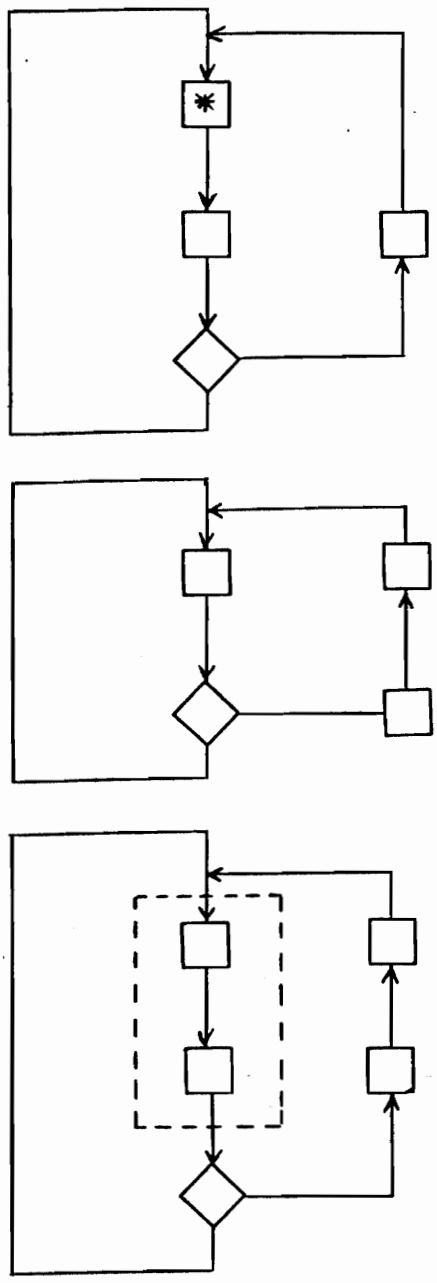


Figure 5. Productions



Derivation of Wirth's Scanning Routine

Figure 6.



Regular

Not Regular

Corrected

The RPC Conditon

Figure 7.

C. Use and Interpretation of the Model.

A Markov Process with n states may be described by a n by n transition matrix containing a probability value in each element p(ij) corresponding to the probability of making a transition from state i to state j. The transition matrix for Wirth's scanning routine (fig. 2) is shown in figure 8a. This matrix must be stochastic; that is, the sum of the elements of each row must be 1. Applying this to figure 8a, we find that p+q=r+s=t+u=1. Taking row 2 of the matrix as an example, note that p and q represent the probability that, after a transition from state 2 has been made, the process will be in states 1 or 3, respectively. Since, after the transition from state 2 has been made the machine must be in state 1 of state 3, the probabilities p and q must sum to 1.

If values are assigned to the probabilities, the first order transition matrix (fig. 8b) is obtained. If this matrix is then raised to the second power, the result is the second order transition matrix (fig. 8c). The matrix element p(ij) of T^2 represents the probability that the process will go from state i to state j in exactly two state transitions. In general, an element p(ij) of a mth order transition matrix (the mth power of T) will represent the probability of the process going from state

i to state j in exactly m transitions.

If the process represented by the transition matrix T is strongly ergodic, the sequence of powers of T will eventually converge to T*, the equilibrium matrix (fig. 8d). This sequence of transition matrices is called the Finite Markov Chain.

The equilibrium matrix of a Finite Markov Chain has the property that all of its rows are equal and that all of its elements are positive (nonzero). This implies that after reaching the equilibrium point, the process will have the probability p(ij) in T* of being in state it was in initially, because, given j, p(ij) in T* is the same for all i.

Since the rows of the equilibrium matrix are equal, the matrix may be represented by a limit vector (fig. 8e). This limit vector is the principal eigenvector of the matrix T. The information in the limit vector may now be used to determine the relative frequency of execution of the flowgraph nodes associated with the process states. From figure 8e we find that nodes 1 and 2 are executed twice as often as nodes 3, 5, and 6, and nearly three times as often as node 4.

Looking at the modelling procedure as explained above, we can now see the purpose of the strongly ergodic condition. If the process could never move from some

state to another, then two disjoint equivalence classes would be formed of the first state together with all states which could go to it, or to which it could go, in a finite number of transitions, and the second state together with all states which could go to it, or to which it could go, in a finite number of transitions. The process would never be able to go from a state in the first class to a state in the second class in any number of transitions. This means that there would be zeros in the corresponding elements of transition matrices of all orders making it impossible for an equilibrium matrix to exist.

As for the rpc condition, let us look at the non-regular flowgraph in figure 7. If the process begins in state 1 at time 1 then it will never be in state 1 at an even (2, 4, 6, ...) time because state 1 has only even cycles and time 1 + an even number is always odd. Thus the transition matrices will not approach equilibrium, but will oscillate. If the rpc condition is not met in a flowgraph, it may easily be corrected by splitting a block in the flowgraph. For example, the non-regular flowgraph in figure 7 may be made regular by splitting a block as in the corrected flowgraph of figure 7.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ p & 0 & q & 0 & 0 & 0 \\ 0 & 0 & 0 & r & s & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ t & 0 & 0 & 0 & u & 0 \end{bmatrix}$$

a. T

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ .5 & 0 & .5 & 0 & 0 & 0 \\ 0 & 0 & 0 & .75 & .25 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ .25 & 0 & 0 & 0 & .75 & 0 \end{bmatrix}$$

b. T^1

$$\begin{bmatrix} .5 & 0 & .5 & 0 & 0 & 0 \\ 0 & .5 & 0 & .37 & .13 & 0 \\ .75 & 0 & 0 & 0 & 0 & .25 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ .25 & 0 & 0 & 0 & .75 & 0 \\ 0 & .25 & 0 & 0 & 0 & .75 \end{bmatrix}$$

c. T^2

$$\begin{bmatrix} .26 & .26 & .13 & .09 & .13 & .13 \\ .26 & .26 & .13 & .09 & .13 & .13 \\ .26 & .26 & .13 & .09 & .13 & .13 \\ .26 & .26 & .13 & .09 & .13 & .13 \\ .26 & .26 & .13 & .09 & .13 & .13 \\ .26 & .26 & .13 & .09 & .13 & .13 \end{bmatrix}$$

d. T^*

$$[.26 \ .26 \ .13 \ .09 \ .13 \ .13]$$

e. Limit Vector

Figure 8

III. The Problem of Validity.

A. The Problem.

Having designed the Markov Model, there is still a problem. That is, how does one know what values to assign to the state transition probabilities for the decision states? The probabilities used in figure 8 were chosen arbitrarily; this raises serious questions as to the validity of the model.

1) Concerning the model, is the predicted behaviour dependent on the probabilities assigned to the transitions as opposed to the structure of the isomorphic flowgraph?

2) Concerning programs, is the behaviour dependent on input data as opposed to the control structure of the program?

3) Whether or not the model and programs are dependent on transition probabilities and input data, does the model accurately predict runtime behaviour?

4) If not, can it be made to do so?

If, as in question 1, we wish to measure the variation of a program's behaviour over all value assignments to the transition probabilities, a system of parameterized equations may be found analytically, however this procedure becomes much more complex as the number of decision nodes in the modelled flowgraph increases. It is because of this that a statistical technique of investigating sample populations of behaviours is chosen instead.

B. The Experiment.

To answer the preceding questions an an experiment is now described comparing the modelled and runtime behaviours of sample programs. To do this need a representation of program behaviour is needed which provides a measure of the similarity of different behaviours. In the following sections, such a representation will be derived, and some tests based on it will be created to establish the validity of the model.

B.1. The Geometric Representation of Program Behaviour.

The behaviour of a program whose flowgraph has n nodes may be represented as a point in n dimensional space. This point is defined by the vector B=(b1,b2,...,bn) where bi is the relative frequency of execution of node i. In fact, the point lies on the n-1 dimensional flat in n-space, since the vector elements are under the additional constraint that their sum must equal one.

As an example, the behaviour of the simple flowgraph with three nodes and a fixed branch probability p in figure 9(a) may be represented by the vector below it and shown as a point on the plane b1+b2+b3=1 (fig. 9b).

With another branch probability q, the behaviour changes and occupies a different point (B') on the plane. To measure this difference the cosine of the angle (theta) formed by the line segments between each point and the origin (fig. 9c) is found by the following formula:

$$\cos(\text{theta}) = \frac{B \bullet B'}{||B|| \bullet ||B'||} ,$$

where $A \bullet B$ is the dot product,

$$\sum_{i=1}^{n} ai \bullet bi \ , \ A=(a1,a2,\ldots,an), \ B=(b1,b2,\ldots,bn) \ ,$$

and $||A||$ is the length, $\sqrt{\sum_{i=1}^{n} (ai)^2}$

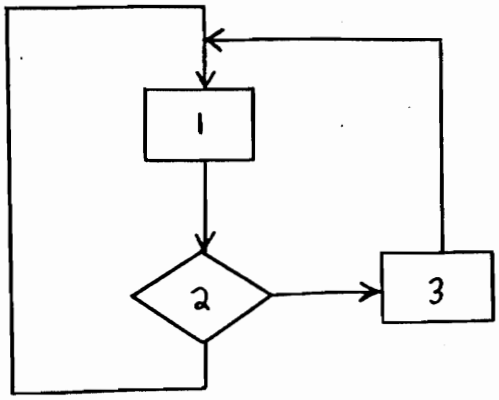
As the behaviours become closer the angle approaches 0° and the cosine approaches 1.

If behaviours of a program with n nodes are modelled for k different value assignments to the branch probabilities, a centroid or "average" behaviour (C) may be found by calculating the mean value of each element in all samples. Thus,

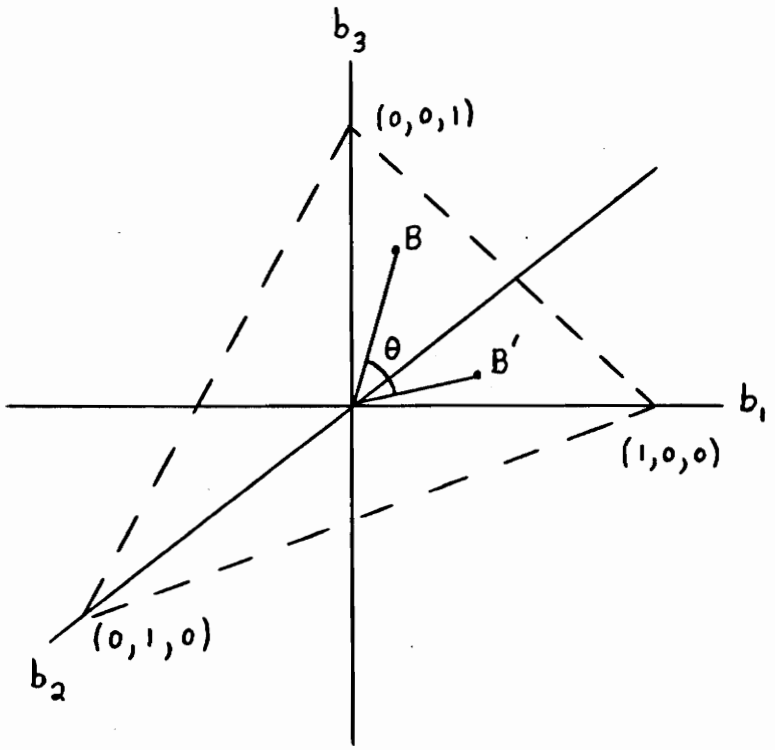
$$C = (c1, c2, \ldots, cn)$$

$$\text{where } ci = \sum_{j=1}^{k} bji/k \quad , \quad bji = \text{ith element of } Bj.$$

Finally the cosine of the angle between each sample behaviour and the centroid is calculated and tabulated on a distribution histogram. This histogram shows the number of behaviours at each distance (expressed as the cosine value) from the centroid. A tight clustering of these values about the cosine 1.0 on the histogram indicates that the behaviour is relatively independent of the factor being varied.



a.



b.

Geometric Representation of Program Behaviour

Figure 9.

B.2. The Tests.

With the geometric representation in mind, we now design the tests which will answer the questions raised in Section A.

The information in the distribution histogram is sufficient to answer the first question as to how dependent modelled behaviour is on the process state transition probabilities, if the factor that is varied to get the distribution is these probabilities. Thus, the first test is to select several programs, model each one for a large number of randomly generated sets of transition probabilities, and examine the resulting behaviour distribution. If all behaviours are tightly grouped around the 1.0 cosine bar of the distribution histogram then the points they represent are clustered closely on the n-1 dimensional flat, and the modelled behaviour is relatively independent of transition probabilities, otherwise not.

Similarly, the second question is answered by running the same sample programs with the same large number of randomly generated sets of data. The relative frequency of execution of code blocks, and thus the behaviour of the program is determined by inserting code into the program to count the execution frequency of each

block then express it as a percentage of the total number of block executions in the run.

A comparison of the model and run time distribution histograms and centroids should adequately answer the third question. The fourth question is deferred until the other results are found.

IV. Results.

Following Halstead's example [2], Algorithms 1 through 14 from the Communications of the ACM [7-20] were chosen as the sample programs (Appendix A). These underwent the minimum restructuring necessary to conform to the structured program and rpc conditions (Appendix B). This required alterations such as duplicating portions of the program, adding flags (boolean variables) to propagate termination conditions, and splitting nodes to meet the rpc condition. A regular flowgraph was constructed and labelled for each program (Appendix C), with two exceptions:

1) Algorithm 9 is composed of a main routine and subroutine, so that a flowgraph (9 and 9a) was constructed for each of these.

2) Algorithms 10 thru 13 are identical in structure, so only one flowgraph was constructed for all of them.

One hundred randomly generated sets of uniformly distributed probability values were assigned to the transition matrix of each flowgraph's model. The limit vectors were obtained by successively squaring the transition matrix, to reduce computing time. They were then used as behaviour points in the creation of a distribution histogram.

From these it is fairly obvious that while the behaviours of several of the algorithms are tightly clustered (2, 4, 6, 7, 8, 10-13, 14), some are not (1, 3, 5, 9, 9a) . The best and worst cases of each are shown in figure 10. The modelled behaviour for the latter structures is spread well over the range of possible behaviours. It is also interesting to note that all of the algorithms with 25 or more nodes in their flowgraph have ill-behaved models.

For the second test, the structured equivalents of algorithms 3, 4, 5, 7, and 10 were recoded in PL/1, because of the control structures available in that language. These algorithms were chosen for ease of data generation from both the well- and ill-behaved algorithms of the first test. Algorithm 10 was chosen to represent Algorithms 10 thru 13, since these algorithms have equivalent conditional expressions for all decision nodes. Code was added to count the number of times each node was executed. Each of these node execution frequencies was divided by the total number of node executions at the end of each run to provide a relative frequency of execution expressed as a percentage of the total number of node executions. In these, the behaviours of Algorithms 3, 4, 7, and 10 cluster very tightly, and those of Algorithm 5 slightly less. The runtime behaviour distribution

histogram is shown in figure 11 for comparison with the worse case in figure 10.

At this point three of our questions have been answered. The behaviour of these programs is much more dependent on their structure than on input data but this is not modelled well by the Markov Model. Looking at the cosines of the angles between the modelled and runtime behaviour centroids (fig. 12) we see that the correspondence is generally poor, and at best fair. This leads us to the last question, is it possible to improve the accuracy of the behaviour prediction by altering the modelling procedure?

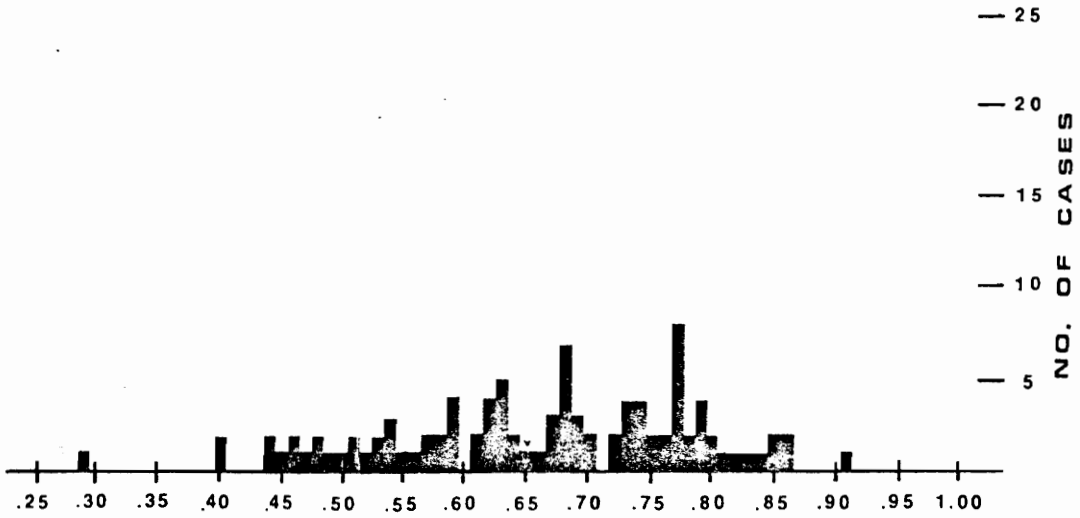
What makes the behaviour predicted from the model so different from the runtime behaviour is the difference in the probabilities assigned to the state transitions from the actual probabilities for the program branches? In the model, these values are varied randomly over the range 0 to 1. If this range is restricted, yet still covers the actual probability of the program branch, then the correspondence between modelled and runtime behaviour should improve. Two assumptions that may reasonably be made about program branching probabilities are:

1) The decision node of a DO loop will continue executing the loop more often than it will exit the loop.

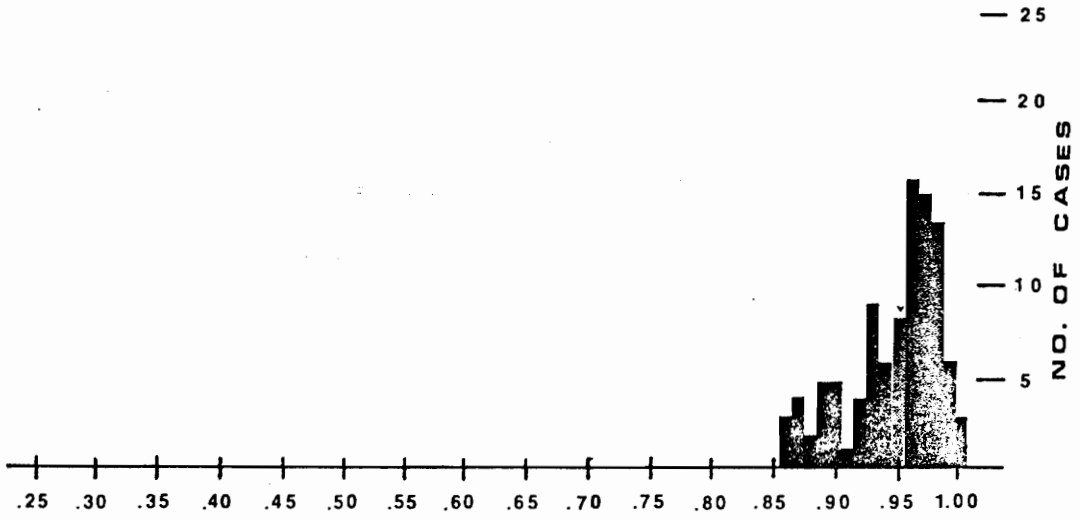
2) Branches directly to termination blocks (stops, returns, or termination flag setting) will occur less frequently than other branches from the same decision node.

The effect of these assumptions is illustrated in figure 13.

Thus, the third test conducted was a repetition of the first test with the difference that the probabilities generated for decision nodes in these situations were restricted to the ranges given in figure 13. The resulting distribution histograms are remarkably closer to the runtime results than are those produced by the original model, as can be seen by comparing figures 10, 12, and 14. This conclusion is well supported by the statistics in figure 12. It can also be seen if the runtime and modified model behaviour centroids are compared. The three or four most frequently executed nodes in the program correspond with those states in the centroid of the modified model with the highest probabilities.



ALGORITHM 3 COSINES OF SAMPLES ABOUT CENTROID



ALGORITHM 4 COSINES OF SAMPLES ABOUT CENTROID

Figure 10.

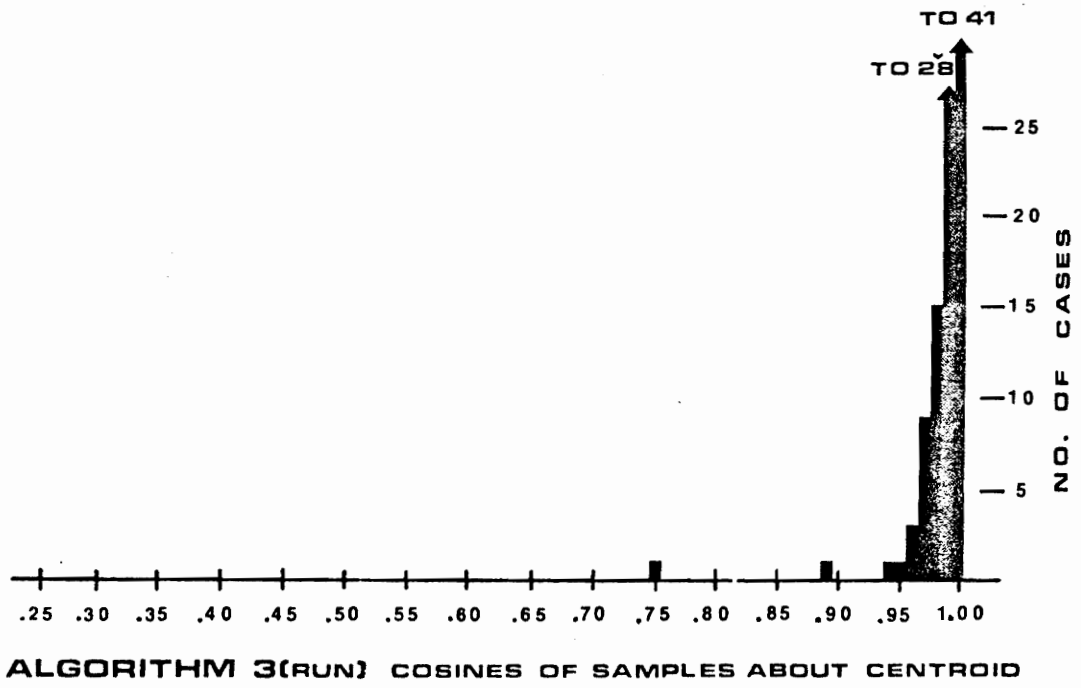
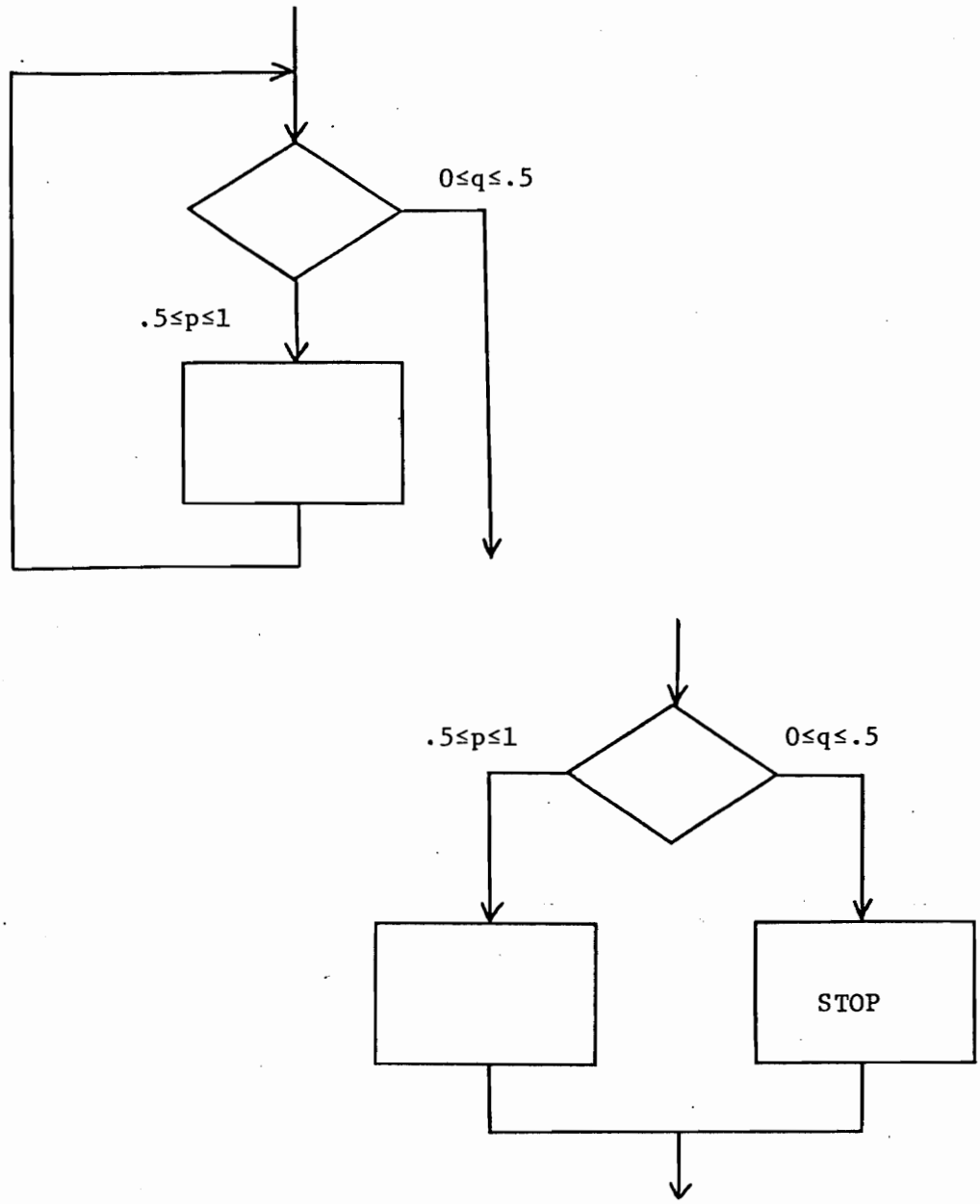

TO 41
TO 28
25
20
15
10
5
NO. OF CASES
.25 .30 .35 .40 .45 .50 .55 .60 .65 .70 .75 .80 .85 .90 .95 1.00
ALGORITHM 3(RUN) COSINES OF SAMPLES ABOUT CENTROID

Figure 11.

Algorithm	Model and Run	Model with Directed Inequalities and Run
3	.835	.948
4	.857	.974
5	.755	.878
7	.548	.983
10	.557	.974

Cosines of Angles between Centroids

Figure 12.



Branch Probility Inequality Directions

in Modified Model

Figure 13.

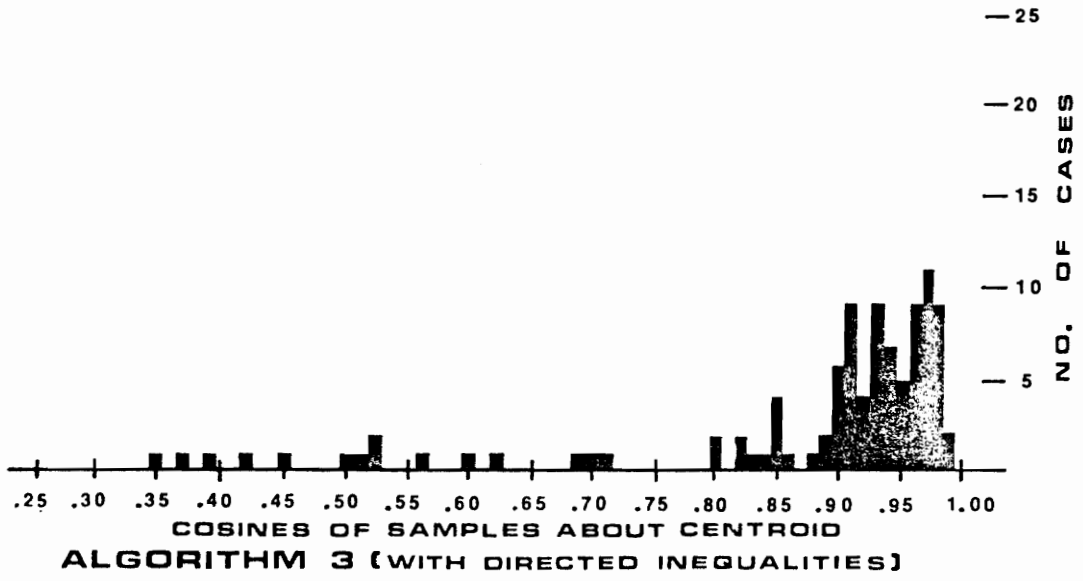

25
20
15
10
5
NO. OF CASES
.25 .30 .35 .40 .45 .50 .55 .60 .65 .70 .75 .80 .85 .90 .95 1.00
COSINES OF SAMPLES ABOUT CENTROID
ALGORITHM 3 (WITH DIRECTED INEQUALITIES)

Figure 14.

V. Conclusions and Recommendations.

In light of the evidence presented, the modified model has been shown to be valid. However, as with any other validation using test cases, the results cannot be considered a truly formal or general proof of its correctness. However they provide grounds for future work attempting to apply the model, which leads to a series of proposals for future endeavours.

Rauscher [1] intended that modelling processors be made an integral part of optimizers in translator systems for dynamically microprogrammable machines. This is slightly more difficult to achieve automatically with the modified model than with the original one, since DO blocks and termination code must be recognized by the processor, however it is still very feasible. This shows that one useful investigation would be to search for other easily recognizable structures which may be used to reduce the range of transition probabilities generated for the model, thus further improving its accuracy.

The modelling process takes a good deal of computing time, so that decreasing the number of sample probability sets generated could tremendously increase the efficiency of the model processor itself with respect to execution time of the optimizer. Further statistical tests should

be devised to determine the number of samples necessary for an accurate result.

Finally, the model is at a stage where techniques should be developed for using it in applications such as optimization by dynamic microprogramming [1], data flow analysis for debugging [21], and storage allocation in operating systems using paging. For all of these applications, knowledge of the relative frequency of code blocks is not enough, other information must be attached to the states of the model before it may applied to the problem. When dealing with the optimization of execution time, the execution time of the code in each block is needed. For data flow analysis and storage allocation purposes, a table of the variables used in each block and their attributes is necessary.

All of this makes the author feel that there is still much work to be done with this Markov Model of Program Behaviour.

VI. References.

1. T. G. Rauscher, "Dynamic Problem Oriented Redefinition of Computer Architecture via Microprogramming", (unpublished Ph.D. thesis) University of Maryland, 1975.

2. M. H. Halstead, "Natural Laws Controlling Algorithm Structure?", SIGPLAN Notices (Feb. 1972) pp. 19-26.

3. T. C. Wessekamper and R. W. Zoladz, "A Markov Model of Certain Structured Programs", VPI&SU Tech. Report #CS76004-R, 1976.

4. G. Bohm, G. Jacopini, "Flow Diagrams, Turing Machines, and Languages with Two Formulation Rules", CACM 9, No. 5 (May 1966) pp. 366-71.

5. E. W. Dijkstra, "Notes on Structured Programming", Part 1 of the book "Structured Programming", C. A. R. Hoare, editor, Academic Press, 1972.

6. J. J. Martin, "The 'Natural' Set of Basic Control Structures", SIGPLAN Notices 8, No. 12 (December 1973) pp. 5-14.

7. R. J. Herbold, Alg. 1, QUADI, CACM 3, No. 2 (Feb. 1960) pp. 74.

8. J. Wegstein, Alg. 2, Rootfinder, CACM 3, No. 2 (Feb. 1960) pp. 74.

9. A. A. Grau, Alg. 3, Solution of Polynomial by Bairstow-Hitchcock Method, CACM 3, No. 2 (Feb. 1960) pp. 74-75.

10. S. Gorn, Alg. 4, Bisection Routine, CACM 3, No. 3 (March 1960) pp. 174.

11. D. S. Clarke, Alg. 5, Bessel Function I, Series Expansion, CACM 3, No. 4 (April 1960) pp. 240.

12. D. S. Clarke, Alg. 6, Bessel Function I, Asymptotic Expansion, CACM 3, No. 4 (April 1960) pp. 240.

13. R. Clausen, Alg. 7, Euclidian Algorithm, CACM 3, No. 4 (April 1960) pp. 240.

14. P. Naur, et al, Alg. 8, Euler Summation, CACM 3, No. 5 (May 1960) pp. 311.

15. P. Naur, et al, Alg. 9, Runge-Kutta Integration, CACM 3, No. 5 (May 1960) pp. 312.

16. G. M. Galler, Alg. 10, Evaluation of the Chebyshev Polynomial Tn(X) by Recursion, CACM 3, No. 6 (June 1960) pp. 353.

17. G. M. Galler, Alg. 11, Evaluation of the Hermite Polynomial Hn(X) by Recursion, CACM 3, No. 6 (June 1960) pp. 353.

18. G. M. Galler, Alg. 12, Evaluation of the Lauguerre Polynomial Ln(X) by Recursion, CACM 3, No. 6 (June 1960) pp. 353.

19. G. M. Galler, Alg. 13, Evaluation of the Legendre Polynomial Pn(X) by Recursion, CACM 3, No. 6 (June 1960) pp. 353.

20. A. Beam, Alg. 14, Complex Exponential Integral, CACM 3, No. 7 (July 1960) pp. 406.

21. L. D. Fosdick, L. J. Osterweil, "Data Flow Analysis in Software Reliability", Computing Surveys 8, No. 3 (Sept. 1976) pp. 305-330.

Appendix A.

Algorithms 1-14

Communications of the ACM

Algorithm 1.

QuadI, R. J. Herbold, CACM 3, 74.

```
procedure  QuadI (a,b,m,n,p,wk,uk,P(B,j)=:(c))=:(Ij)
           begin
QuadI:                h:=(b-a)/m
           for        j:=1(1)p; Ij:=0
                      A:=a-h/2
           for        i:=1(1)m
L1:        begin      A:=A+h
           for        k:=1(1)n
L2:        begin      B:=A+(h/2)*uk
           for        j:=1(1)p
L3:        begin      P(B,j)=:(c)
                      Ij:=Ij+wk*c     end L3 ;  end L2
                      end L1
           for        j:=1(1)p
                      Ij:=(h/2)*Ij
           return
           integer    (j,k,i)
           end        QuadI
```

Algorithm 2.

Rootfinder, J. Wegstein, CACM 3, 74.

```
procedure   Root(f(),a,eps)=:(g)
begin
Root:       b:=a ; c:=f(b) ; g:=c
            if (c=a) ; return
            d:=a ; b:=c ; e:=c
Hob:        c:=f(b)
            g:=(d*c-b*e)/(c-e-b+d)
            if (abs((g-b)/g)≤eps) ; return
            e:=c ; d:=b ; b:=g ; go to Hob
end
```

Algorithm 3.

Solution of Polynomial Equation by Bairstow-Hitchcock Method

A. A. Grau, CACM 3, 74-5.

```
procedure
BAIRSTOW    (n,a[ ],eps0,eps1,eps2,eps3,K)=:
            (m,x[ ],y[ ],nat[ ],ex[ ]);
begin
integer     (i,j,k,n1,n2,m1) ;
array       (b,c[0:n+1]) ;
BAIRSTOW    for i:=0(1)n ; b[i]:=a[i]
            b[n+1]:=0 ; n2:=entire((n+1)/2)
             n1:=2*n2
            for m1:=1(1)n2 ; begin p:=0 ; q:=0
            for k:=1(1)K ; begin
step:       for i:=0(1)n1 ; c[i]:=b[i]
            for j:=n1-2,n1-4 ; begin
            for i:=0(1)j ; begin
            c[i+1]:=c[i+1]-p*c[i]
            c[i+2]:=c[i+2]-q*c[i] end end
            r0:=c[n1] ; r1:=c[n1-1]
            s0:=c[n1-2] ; s1:=c[n1-3]
            v0:=-q*s1 ; v1:=s0-s1*p
            det0:=v1*s0-v0*s1
            if (abs(det0)<eps0) ; begin
            p:=p+1 ; q:=q+1 ; go to step end
            det1:=s0*r1-s1*r0
            det2:=r0*v1-v0*r1
            incrp:=det1/det0 ; incrq:=det2/det0
            p:=p+incrp ; q:=q+incrq
            if (abs(r0)<eps1) ; begin
            if (abs(r1)<eps1) ; begin
            ex[m1]:=1 ; go to next end end
            if (abs(incrp)<eps2) ; begin
            if (abs(incrq)<eps2) ; begin
            ex[m1]:=2 ; go to next end end
            if (abs(incrp/p)<eps3) ; begin
            if (abs(incrq/q)<eps3) ; begin
            ex[m1]:=3 ; go to next end end
            ex[m1]:=4
```

```
next:       S:=-p/2 ; T:=S²-q
            if (T≥0) ; begin T:=sqrt(T)
            nat[m1]:=1 ; x[m1]:=S+T
             y[m1]:=S-T end
            if (T<0) ; begin nat[m1]:=-1 ; x[m1]:=S
             y[m1]:=sqrt(-T) end
            if (ex[m1]=4) ; go to out
            for j:=0(1)(n1-2) ; begin
            b[j+1]:=b[j+1]-p*b[j]
            b[j+2]:=b[j+2]-q*b[j] ; end
            n1:=n1-2 ; if (n1<1)
out:        begin m:=m1 ; return end
            if (n1<3) ; begin
            m1:=m1+1 ; ex[m1]:=1
            p:=b[1]/b[0] ; q:=b[2]/b[0]
            go to next end
end end
```

Algorithm 4.

Bisection Routine.

S. Gorn, CACM 3, 174.

```
procedure   Bisec(y1,y2,e,e1,F(),FLSXT)=:(x)
begin
Bisec:      i:=1 ; j:=1 ; k:=1 ; x:=y2
a:          f:=F(x) ; if (abs(f)≤e) ; return
            go to g[i]
First val:  i:=2 ; f1:=f ; x:=y1 ; go to a
Succ val:   if (sign(f)=sign(f1)) ; go to d[j]
            go to n[k]
Sec val:    j:=2 ; k:=2
Midpoint:   x:=y1/2+y2/2 ; go to a
Reg d:      y2:=x
Precision:  if (abs(y1-y2)≥e1) ; go to Midpoint
            return
Reg n:      y1:=x ; go to Precision
            integer (i,j,k)
            switch g:=(First val,Succ val)
            switch d:=(FLSXT,Reg d)
            switch n:=(Sec val,Reg n)
end Bisec
```

Algorithm 5.

Bessel Function I, Series Expansion

D. S. Clarke, CACM 3, 240.

```
procedure   I(n,X,d)=:(Is)
begin
I:          s:=0 ; sum:=0
if          (n≠0) ; go to STRT
if          (X=0) ; begin Is:=1 ; return end
            summ:=1 ; go to SURE
STRT:       sfac:=1
if          (s=0) ; go to HRE
for         t:=1(1)s
            sfac:=sfac*t
HRE:        snfac:=sfac
for         t:=s+1(1)s+n
            snfac:=snfac*t
            summ:=sum+(X/2)**(n+2*s)/(sfac*snfac)
SURE: if    (d<abs(summ-sum))
begin       s:=s+1 ; sum:=summ ; go to STRT end
            Is:=summ ; return
end
```

Algorithm 6.

Bessel Function I, Asymptotic Expansion

D. S. Clarke, CACM 3, 240.

```
procedure    I(n,X,d)=:(IA)
begin
I:           r:=1 ; pe:=(4*n²-1)/(8*X)
               sum:=-pe
Repeat:      r:=r+1
             pe:=pe*((2*n)²-(2*r-1)²)/(r*8*X)
if           (d<abs(pe))
begin        sum:=sum+(-1)**r*pe ; go to Repeat end
             IA:=(1+sum)*(exp(X)/sqrt(2*pi*X))
              return
end
```

Algorithm 7.

Euclidian Algorithm

R. Claussen, CACM 3, 240.

```
procedure   EUC (a,b)=:(gcd)
begin
EUC:
if          (a=0)
begin       gcd:=b ; return end
if          (b=0)
begin       gcd:=a ; return end
            r2:=a
            r1:=b
here:       g:=r2/r1
comment     Assumption is made that truncation takes
            place in the above statement;
            r:=r2-r1*g
if          (r=0)
begin       gcd:=r1 ; return end
begin       r2:=r1
            r1:=r
            go to here end
integer     (g)
end
```

Algorithm 8.

Euler Summation

P. Naur, et al, CACM 3, 311.

```
procedure euler (fct, sum, eps, tim) ;
value eps, tim ;
integer tim ; real procedure fct ;
real sum, eps ;
begin integer i,k,n,t ; array m[0:15] ;
real mn,mp,ds ;
i:=n:=t:=0 ; m[0]:=fct(0) ; sum:=m[0]/2 ;
nextterm: i:=i+1 ; mn:=fct(i) ;
          for k:=0 step 1 until n do
              begin mp:= (mn+m[k])/2 ; m[k] := mn ;
                mn :=mp end means ;
          if (abs(mn)<abs(m[n])) or (n<15) then
              begin ds:=mn/2 ; n:=n+1 ; m[n]:=
                mn end accept
          else ds:=mn ;
          sum:=sum+ds ;
          if abs(ds)<eps then t:=t+1 else t:=0 ;
          if t<tim then go to nextterm
end euler
```

Algorithm 9.

Runge-Kutta Integration

P. Naur, et al, CACM 3, 312.

```
procedure   RK(x,y,n,FKT,eps,eta,xE,yE,fi) ;
value  x,y ; integer  n ; Boolean  fi ;
real  x,eps,eta,xE ; array y,yE ; procedure  FKT ;
begin
 array  z,y1,y2,y3[1:n] ; real  x1,x2,x3,H ;
 Boolean out ; integer k,j ; own real s,Hs ;
 procedure   RK1ST(x,y,h,xe,ye) ;
  real  x,h,xe ; array  y,ye ;
  begin
   array  w[1:n], a[1:5] ; integer j,k ;
   a[1]:=a[2]:=a[5]:=h/2 ; a[3]:=a[4]:=h ;
   xe:=x ;
   for k:= 1 step 1 until n do ye[k]:=w[k]:=y[k] ;
   for j:= 1 step 1 until 4 do
   begin
    FKT(xe,w,n,z) ;
    xe:=x+a[j] ;
    for k:=1 step 1 until n do
    begin
     w[k]:=y[k]+a[j]*z[k] ;
     ye[k]:=ye[k]+a[j+1]*z[k]/3 ;
    end k
   end j
 end RK1ST ;
```

```
Begin of program:
    if fi then begin H:=xE-x ; s:=0 end else H:=Hs ;
    out:=false ;
AA: if (x+2.01*H-xE>0)=(H>0) then
    begin Hs:=H ; out:=true ; H:=(xE-x)/2
     end if ;
    RK1ST(x,y,2*H,x1,y1);
BB: RK1ST(x,y,H,x2,y2) ; RK1ST(x2,y2,H,x3.y3) ;
    for k:=1 step 1 until n do
        if comp(y1[k],y3[k],eta)>eps then go to CC ;
    comment : comp(a,b,c) is a function designator,
     the value of which is the absolute value of the
     difference of the mantissae of a and b, after
     the exponents of these quantities have been
     made equal to the largest of the exponents of
     the originally given parameters a,b,c ;
    x:=x3 ; if out then go to DD ;
    for k:=1 step 1 until n do y[k]:=y3[k] ;
    if s=5 then begin s:=0 ; H:=2*H end if
    s:=s+1 ; go to AA ;
CC: H:=0.5*H ; out:=false ; x1:=x2 ;
    for k:=1 step 1 until n do y1[k]:=y2[k] ;
    go to BB ;
DD: for k:=1 step 1 step 1 until n do yE[k]:=y3[k]
end RK
```

Algorithm 10.

Evaluation of the Chebyshev Polynomial Tn(X)

by Recursion

G. M. Galler, CACM 3, 353.

```
real procedure  Ch(n,X) ;
real            X ; integer n ;
begin real      a,b,c ; integer i ;
                a:=1 ; b:=X ;
                if n=0 then c:=a else if n=1 then
                c:=b else for i:=2 step 1 until n do
                begin c:=2*X*b-a ;
                      a:=b ; b:=c
                end
                Ch:=c
                end
```

Algorithm 11.

Evaluation of the Hermite Polynomial Hn(X)

by Recursion

G. M. Galler, CACM 3, 353.

```
real procedure  He(n,X) ;
real            X ; integer n ;
begin real      a,b,c ; integer i ;
                a:=1 ; b:=2*X ;
                if n=0 then c:=a else if n=1 then
                c:=b else for i:=1 step 1 until n-1 do
                begin c:=2*X*b-2*i*a ;
                      a:=b ; b:=c
                end
                He:=c
                end
```

Algorithm 12.

Evaluation of the Laguerre Polynomial Ln(X)

by Recursion

G. M. Galler, CACM 3, 353.

```
real procedure  La(n,X) ;
real            X ; integer n ;
begin real      a,b,c ; integer i ;
                a:=1 ; b:=1-X ;
                if n=0 then c:=a else if n=1 then
                c:=b else for i:=1 step 1 until n-1 do
                begin c:=(1+2*i-X)*b-(i**2)*a ;
                      a:=b ; b:=c
                end
                La:=c
                end
```

Algorithm 13.

Evaluation of the Legendre Polynomial Pn(X) by Recursion

G. M. Galler, CACM 3, 353.

```
real procedure  Le(n,X) ;
real            X ; integer n ;
begin real      a,b,c ; integer i ;
                a:=1 ; b:=X ;
                if n=0 then c:=a else if n=1 then
                c:=b else for i:=1 step 1 until n-1 do
                begin c:=b*X+(i/(i+1))*(X*b-a) ;
                      a:=b ; b:=c
                end
                Le:=c
                end
```

Algorithm 14.

Complex Exponential Integral

A. Beam, CACM 3, 406.

```
procedure   EKZ(x,y,k,e,u,v,n) ; real x,y,k,e,u,v ;
             integer n ;
begin        real t1,t2,t3,M,K,c,a,d,b,g,h,e1 ;
             integer m ;
             e1:=e**2 ;
            u:=c:=a:=1 ; v:=d:=b:=0 ;
             n:=1 ; K:=k-1 ;
BACK:       g:=u ; h:=v ; n:=n+1 ;
             m:=n/2 ;
            if 2*m=n then M:=m+K else M:=m ;
            t1:=x+M*c ; t2:=y+M*d ;
             t3:=t1**2+t2**2 ;
            c:=(x*t1+y*t2)/t3 ;
             d:=(y*t1-x*t2)/t3 ;
            t1:=c-1 ; t2:=a ;
             a:=a*t1-d*b ; b:=d*t2+t1*b ;
            u:=g+a ; v:=h+b ;
            if (a**2+b**2)/(u**2+v**2)>e1 then go to
             BACK ;
end         EKZ
```

Appendix B.

Algorithms 1-14.

Restructured Code

Algorithm 1.

```
QUADI: PROC (A,B,M,N,P,W,U,P)
       RETURNS ((*) FLOAT DEC);
DCL (A,B) FLOAT DEC, (M,N,P) FIXED BIN;
DCL 1 GAUSS_VAL CONTROLLED,
      2 W(1:N) FLOAT DEC,
      2 U(1:N) FLOAT DEC,
      2 II(1:P) FLOAT DEC;
DCL (H,AA,BB) FLOAT DEC, (I,J,K) FIXED BIN;
DCL PP ENTRY (FLOAT DEC, FIXED BIN)
       RETURNS(FLOAT DEC);
H=(B-A)/M;
DO J=1 TO P;
   II(J)=0;
END;
AA=A-H/2;
L1: DO I=1 TO M;
       AA=AA+H;
   L2: DO K=1 TO N;
          BB=AA+(H/2)*U(K);
      L3: DO J=1 TO P;
             II(J)=II(J)+W(K)*PP(BB,J);
          END L3;
       END L2;
    END L1;
    DO J=1 TO P;
      II(J)=(H/2)*II(J);
    END;
END QUADI;
```

Algorithm 2.

```
ROOT: PROC (FNUM,A,EP,N) RETURNS (FLOAT DEC);
 DCL (A,B,C,D,E,G,EP) FLOAT DEC,
     (N,K,FNUM) FIXED BIN,
     F ENTRY (FLOAT DEC) RETURNS (FLOAT DEC);
 B=A;
 C=F(B);
 G=C;
 IF C=A
    THEN ;
    ELSE DO; D=A;
             B=C;
             E=C;
             K=1;
         HOB: C=F(B);
              G=(D*C-B*E)/(C-E-B+D);
              IF (ABS((G-B)/G)>EP & K<=N)
                 THEN DO; E=C;
                          D=B;
                          B=G;
                          K=K+1;
                          GO TO HOB;
                       END;
         END;
  RETURN(G);
  END ROOT;
```

Algorithm 3.

```
BAIRSTOW: PROC (N,A,EPS0,EPS1,EPS2,EPS3,KK,
                M,X,Y,NAT,EX);
DCL (N,KK) FIXED BIN,
    (EPS0,EPS1,EPS2,EPS3) FLOAT DEC,
    (M,NAT(1:10),EX(1:10)) FIXED BIN,
    (X(1:10),Y(1:10)) FLOAT DEC;
DCL A(0:N) FLOAT DEC;
DCL (I,J,K,N1,N2,M1) FIXED BIN,
    (S,T,P,Q,INCRP,INCRQ,DET0,DET1,
     DET2,R0,R1,S0,S1,V0,V1) FLOAT DEC,
    (B(0:N+1),C(0:N+1)) FLOAT DEC;
DCL (STOPF,STOPF4) BIT(1) INIT('0'B);
STOPF='0'B;
I=0;
DO WHILE (I<=N);
   B(I)=A(I);
   I=I+1;
END;
B(N+1)=0;
N2=TRUNC((N+1.0)/2.0);
N1=2*N2;
M1=1;
DO WHILE (M1<=N2 & STOPF);
   P=0;
   Q=0;
   K=1;
   DO WHILE (K<=KK & STOPF4);
STEP: I=0;
      DO WHILE (I<=N1);
         C(I)=B(I);
         I=I+1;
      END;
      J=N1-2;
      DO WHILE (J>=N1-4);
         I=0;
         DO WHILE (I<=J);
            C(I+1)=C(I+1)-P*C(I);
            C(I+2)=C(I+2)-Q*C(I);
            I=I+1;
         END;
         J=J-2;
      END;
      R0=C(N1);
      R1=C(N1-1);
      S0=C(N1-2);
      S1=C(N1-3);
```

```
      V0=-Q*S1;
      V1=S0-S1*P;
      DET0=V1*S0-V0*S1;
      IF (ABS(DET0)<EPS0)
         THEN DO; P=P+1;
                  Q=Q+1;
                  GO TO STEP;
              END;
      DET1=S0*R1-S1*R0;
      DET2=R0*V1-V0*R1;
      INCRP=DET1/DET0;
      INCRQ=DET2/DET0;
      P=P+INCRP;
      Q=Q+INCRQ;
      IF (ABS(R0)<EPS1 & ABS(R1)<EPS1)
      THEN DO; EX(M1)=1;
               STOPF4='1'B;
           END;
      ELSE IF (ABS(INCRP)<EPS2 & ABS(INCRQ)<EPS2)
           THEN DO; EX(M1)=2;
                    STOPF4='1'B;
                END;
           ELSE IF P>0 & Q>0
                THEN IF (ABS(INCRP/P)<EPS3
                         & ABS(INCRQ/Q)<EPS3)
                     THEN DO; EX(M1)=3;
                              STOPF4='1'B;
                          END;
                     ELSE K=K+1;
   END;
   IF STOPF4
      THEN STOPF4='0'B;
      ELSE EX(M1)=4;
NEXT: S=-P/2;
      T=S**2-Q;
      IF T<0
         THEN DO; NAT(M1)=-1;
                  X(M1)=S;
                  Y(M1)=SQRT(-T);
              END;
         ELSE DO; T=SQRT(T);
                  NAT(M1)=1;
                  X(M1)=S+T;
                  Y(M1)=S-T;
              END;
      IF EX(M1)=4
         THEN DO; M=M1;
                  STOPF='1'B;
              END;
         ELSE DO; J=0;
```

```
                    DO WHILE (J<=N1-2);
                       B(J+1)=B(J+1)-P*B(J);
                       B(J+2)=B(J+2)-Q*B(J);
                       J=J+1;
                    END;
                    N1=N1-2;
                    IF N1<1
                       THEN DO; M=M1;
                                STOPF='1'B;
                            END;
                END;
        IF N1<3 & STOPF
           THEN DO; M1=M1+1;
                    EX(M1)=1;
                    P=B(1)/B(0);
                    Q=B(2)/B(0);
                    GO TO NEXT;
                END;
        IF STOPF
           THEN M1=M1+1;
END;
RETURN;
END BAIRSTOW;
```

Algorithm 4.

```
BISEC: PROC (Y1,Y2,E,E1,FNUM,FLSXT,X);
  DCL (Y1,Y2,E,E1,X) FLOAT DEC,
      FNUM FIXED BIN,
      FLSXT BIT(1), STOPF BIT(1) INIT('0'B);
  DCL FF ENTRY (FLOAT DEC)
         RETURNS (FLOAT DEC);
  DCL (F,F1) FLOAT DEC, (I,J,K) FIXED BIN;
  I=1;
  J=1;
  K=1;
  X=Y2;
ALPHA: F=FF(X);
       IF ABS(F)<=E
          THEN STOPF='1'B;
       IF I=1 & STOPF
          THEN DO; I=2;
                   F1=F;
                   X=Y1;
                   GO TO ALPHA;
               END;
       IF STOPF
       THEN IF SIGN(F)=SIGN(F1)
            THEN DO; IF J=1
                     THEN DO; FLSXT='1'B;
                              STOPF='1'B;
                          END;
                     ELSE DO; Y2=X;
                              IF ABS(Y1-Y2)<E1
                              THEN STOPF='1'B;
                          END;
                 END;
            ELSE DO; IF K=1
                     THEN DO; J=2;
                              K=2;
                          END;
                     ELSE DO; Y1=X;
                              IF ABS(Y1-Y2)<E1
                              THEN STOPF='1'B;
                          END;
                 END;
       IF STOPF
          THEN DO; X=Y1/2+Y2/2;
                   GO TO ALPHA;
               END;
   RETURN;
    END BISEC;
```

Algorithm 5.

```
I: PROC(N,X,DEL);
   DCL (X,DEL) FLOAT DEC, N FIXED BIN;
   DCL (SUM,SUMM,SFAC,SNFAC) FLOAT DEC,
       (S,T) FIXED BIN,
       STOPF BIT(1) INIT('0'B);
   S=0;
   SUM=0;
   IF N≠0
   THEN DO; SFAC=1;
            IF S=0
            THEN ;
            ELSE DO; T=1;
                     DO WHILE(T<=S);
                        SFAC=SFAC*FLOAT(T);
                        T=T+1;
                     END;
                 END;
            SNFAC=SFAC;
            T=S+1;
            DO WHILE(T<=S+N);
               SNFAC=SNFAC*FLOAT(T);
               T=T+1;
            END;
   SUMM=SUM+(-1)**S*(X/2)**FLOAT(N+2*S)/SFAC*SNFAC;
        END;
   ELSE IF X=0
        THEN DO; IS=1;
                 STOPF='1'B;
             END;
        ELSE SUMM=1;
   DO WHILE(DEL<ABS(SUMM-SUM) & STOPF);
      S=S+1;
      SUM=SUMM;
      SFAC=1;
      IF S=0
      THEN ;
      ELSE DO; T=1;
               DO WHILE(T<=S);
                  SFAC=SFAC*FLOAT(T);
                  T=T+1;
               END;
           END;
      SNFAC=SFAC;
      T=S+1;
      DO WHILE(T<=S+N);
         SNFAC=SNFAC*FLOAT(T);
```

```
         T=T+1;
      END;
      SUMM=SUM+(-1)**S*(X/2)**FLOAT(N+2*S)/SFAC*SNFAC;
   END;
   IF STOPF
      THEN IS=SUMM;
   RETURN;
END I;
```

Algorithm 6.

```
I: PROC(N,X,DEL);
   DCL (X,DEL) FLOAT DEC, N FIXED BIN;
   DCL (PE,SUM) FLOAT DEC, R FIXED BIN;
   R=1;
   PE=(4*N**2-1)/(8*X);
   SUM=-PE;
REPEAT: R=R+1;
        PE=PE*(2*N)**2-(2*R-1)**2/(R*8*X);
        IF DEL<ABS(PE)
           THEN DO; SUM=SUM+(-1)**R*PE;
                    GO TO REPEAT;
                END;
   IA=(1+SUM)*(EXP(X)/SQRT(2*3.141592653589793*X));
   RETURN;
  END I;
```

Algorithm 7.

```
EUC: PROC (A,B);
  DCL (A,B,R1,R2,G) FIXED BIN;
  IF A=0
     THEN GCD=B;
     ELSE IF B=0
             THEN GCD=A;
             ELSE DO; R2=A;
                      R1=B;
                      G=R2/R1;
                      R=R2-R1*G;
                      DO WHILE (R≠0);
                         R2=R1;
                         R1=R;
                         G=R2/R1;
                         R=R2-R1*G;
                      END;
                      GCD=R1;
                  END;
  RETURN;
  END EUC;
```

Algorithm 8.

```
 EULER: PROC(FNUM, SUM, EPS, TIM);
  DCL (FNUM,TIM) FIXED BIN,
      (SUM,EPS) FLOAT DEC;
  DCL FCT ENTRY (FIXED BIN, FIXED BIN)
          RETURNS (FLOAT DEC);
  DCL (I,K,N,T) FIXED BIN,
      (M(0:15),MN,MP,DS) FLOAT DEC;
  I=0;
  N=0;
  T=0;
  M(0)=FCT(FNUM,0);
  SUM=M(0)/2;
NEXTTERM: I=I+1;
          MN=FCT(FNUM,I);
          K=0;
          DO WHILE(K<=N);
             MP=(MN+M(K))/2;
             M(K)=MN;
             MN=MP;
             K=K+1;
          END;
          IF (ABS(MN)<ABS(M(N))) & (N<15)
             THEN DO; DS=MN/2;
                      N=N+1;
                      M(N)=MN;
                  END;
             ELSE DS=MN;
          SUM=SUM+DS;
          IF (ABS(DS)<EPS)
             THEN T=T+1;
             ELSE T=0;
          IF (T<TIM)
             THEN GO TO NEXTTERM;
  RETURN;
 END EULER;
```

Algorithm 9.

```
 RK: PROC (X,Y,N,FNUM,EPS,ETA,XE,YE,FI);
  DCL RK1ST ENTRY (FLOAT DEC, (*) FLOAT DEC,
                   FLOAT DEC, FLOAT DEC,
                   (*) FLOAT DEC);
  DCL (X,EPS,ETA,XE) FLOAT DEC, Y(*) FLOAT DEC,
      YE(*) FLOAT DEC, FI BIT(1),
      (N,FNUM) FIXED BIN;
  DCL (X1,X2,X3,H,EN) FLOAT DEC,
      (Z,Y1,Y2,Y3) (1:N) FLOAT DEC,
      (S,HS) FLOAT DEC,
      (K,J) FIXED BIN, OUT BIT(1);
  IF (FI)
     THEN DO; H=XE-X;
              S=0;
          END;
     ELSE H=HS;
  OUT='0'B;
AA: IF BOOL( (X+2.01*H-XE>0),(H>0), '1001'B)
       THEN DO; HS=H;
                OUT='1'B;
                H=(XE-X)/2;
            END;
    CALL RK1ST(X,Y,2*H,X1,Y1);
    CALL RK1ST(X,Y,H,X2,Y2);
    CALL RK1ST(X2,Y2,H,X3,Y3);
    K=1;
    DO WHILE (K<=N);
       EN=TRUNC(LOG10(MAX(ABS(Y1(K)),
                ABS(Y3(K)),ABS(ETA))));
       IF ABS(Y1(K)*10.**(EN-TRUNC(LOG10(Y1(K))))-
              Y3(K)*10.**(EN-TRUNC(LOG10(Y3(K)))))
          >EPS
          THEN DO; H=.5*H;
                   OUT='0'B;
                   X1=X2;
                   K=1;
                   DO WHILE(K<=N);
                      Y1(K)=Y2(K);
                      K=K+1;
                   END;
                   CALL RK1ST(X,Y,H,X2,Y2);
                   CALL RK1ST(X2,Y2,H,X3,Y3);
                   K=1;
               END;
          ELSE K=K+1;
    END;
```

```
X=X3;
IF OUT
   THEN DO; K=1;
            DO WHILE (K<=N);
               YE(K)=Y3(K);
               K=K+1;
            END;
        END;
   ELSE DO; K=1;
            DO WHILE(K<=N);
               Y(K)=Y3(K);
               K=K+1;
            END;
            IF S=5
               THEN DO; S=0;
                        H=2*H;
                    END;
            S=S+1;
            GO TO AA;
        END;
RETURN;
```

```
 RK1ST: PROC(X,Y,H,XE,YE);
  DCL (X,H,XE) FLOAT DEC, (Y,YE) (*) FLOAT DEC;
  DCL A(1:5) FLOAT DEC, (W,Z) (1:N) FLOAT DEC;
  DCL FKT ENTRY(FIXED BIN, FLOAT DEC,
                (*) FLOAT DEC, FIXED BIN,
                (*) FLOAT DEC);
  A(1),A(2),A(5)=H/2;
  A(3),A(4)=H;
  XE=X;
  K=1;
  DO WHILE(K<=N);
     YE(K)=Y(K);
     W(K)=Y(K);
     K=K+1;
  END;
  J=1;
  DO WHILE(J<=4);
     CALL FKT(FNUM,XE,W,N,Z);
     XE=X+A(J);
     K=1;
     DO WHILE(K<=N);
        W(K)=Y(K)+A(J)*Z(K);
        YE(K)=YE(K)+A(J+1)*Z(K)/3;
        K=K+1;
     END;
     J=J+1;
  END;
  RETURN;
 END RK1ST;
END RK;
```

Algorithm 10.

```
CH: PROC (N,X) RETURNS (FLOAT DEC);
 DCL N FIXED BIN, X FLOAT DEC;
 DCL (A,B,C) FLOAT DEC, I FIXED BIN;
 A=1;
 B=X;
 IF (N=0)
    THEN C=A;
    ELSE IF (N=1)
             THEN C=B;
             ELSE DO; I=2;
                      DO WHILE (I<=N);
                         C=2*X*B-A;
                         A=B;
                         B=C;
                         I=I+1;
                      END;
                  END;
 RETURN(C);
END CH;
```

Algorithm 11.

```
HE: PROC (N,X) RETURNS (FLOAT DEC);
 DCL N FIXED BIN, X FLOAT DEC;
 DCL (A,B,C) FLOAT DEC, I FIXED BIN;
 A=1;
 B=2*X;
 IF (N=0)
    THEN C=A;
    ELSE IF (N=1)
            THEN C=B;
            ELSE DO; I=1;
                     DO WHILE (I<=N);
                        C=2*X*B*-2*I*A;
                        A=B;
                        B=C;
                        I=I+1;
                     END;
                 END;
 RETURN(C);
END HE;
```

Algorithm 12.

```
LA: PROC (N,X) RETURNS (FLOAT DEC);
 DCL N FIXED BIN, X FLOAT DEC;
 DCL (A,B,C) FLOAT DEC, I FIXED BIN;
 A=1;
 B=1-X;
 IF (N=0)
    THEN C=A;
    ELSE IF (N=1)
             THEN C=B;
             ELSE DO; I=1;
                      DO WHILE (I<=N);
                         C=(1+2*I-X)*B-I*I*A;
                         A=B;
                         B=C;
                         I=I+1;
                      END;
                  END;
 RETURN(C);
END LA;
```

Algorithm 13.

```
LE: PROC (N,X) RETURNS (FLOAT DEC);
 DCL N FIXED BIN, X FLOAT DEC;
 DCL (A,B,C) FLOAT DEC, I FIXED BIN;
 A=1;
 B=X;
 IF (N=0)
    THEN C=A;
    ELSE IF (N=1)
             THEN C=B;
             ELSE DO; I=1;
                      DO WHILE (I<=N);
                         C=B*X+(I/(I+1))*(X*B-A);
                         A=B;
                         B=C;
                         I=I+1;
                      END;
                  END;
 RETURN(C);
END LE;
```

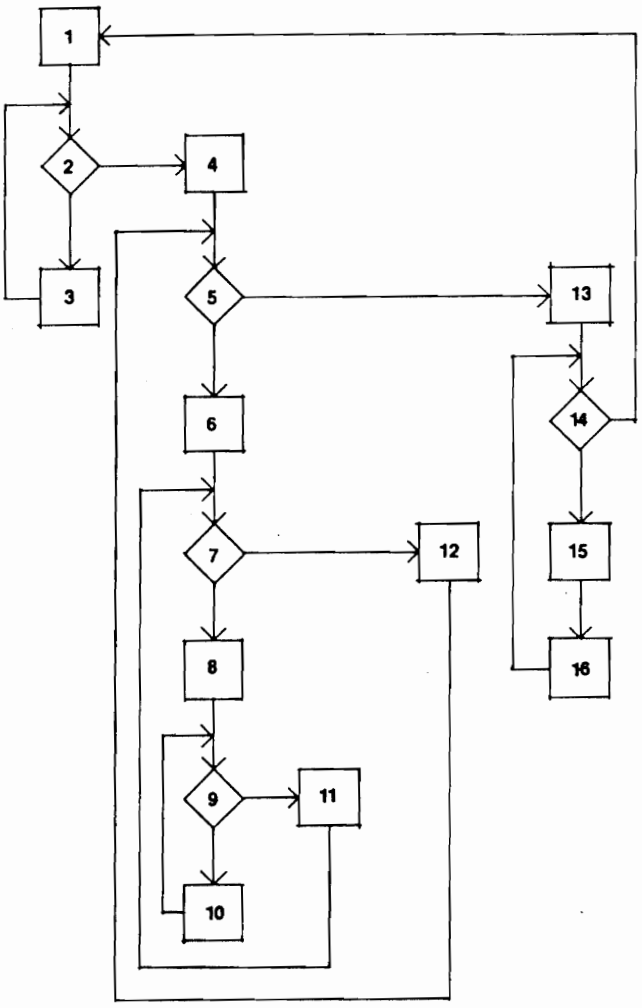
Algorithm 14.

```
EKZ: PROC (X,Y,E,U,V,N);
DCL (X,Y,K,E,U,V) FLOAT DEC, N FIXED BIN;
 DCL (T1,T2,T3,MM,KK,C,A,D,B,G,H,E1) FLOAT DEC,
      M FIXED BIN;
 E1=E**2;
 U=1;
 C=1;
 A=1;
 V=0;
 D=0;
 B=0;
 N=1;
 KK=K-1;
BACK: G=U;
      H=V;
      N=N+1;
      M=N/2;
      IF 2*M=N
         THEN MM=M+KK;
         ELSE MM=M;
      T1=X+MM*C;
      T2=Y+MM*D;
      T3=T1**2+T2**2;
      C=(X*T1+Y*T2)/T3;
      D=(Y*T1-X*T2)/T3;
      T1=C-1;
      T2=A;
      A=A*T1-D*B;
      B=D*T2+T1*B;
      U=G+A;
      V=H+B;
      IF ((A**2+B**2)/(U**2+V**2)>E1)
         THEN GO TO BACK;
END EKZ;
```

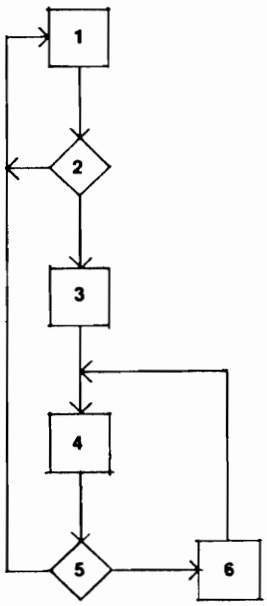
Appendix C.

Algorithms 1-14

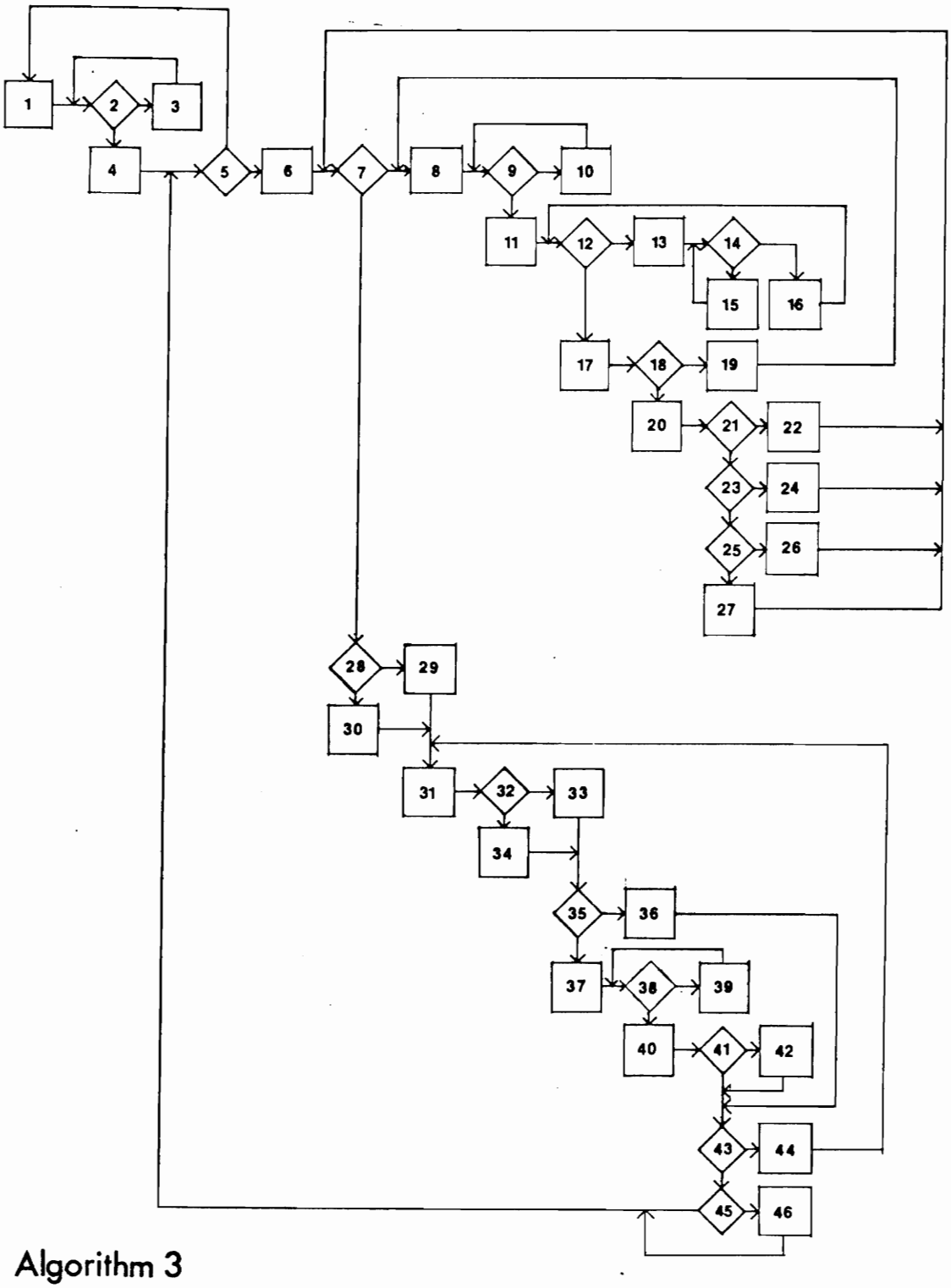
Restructured Flowgraphs



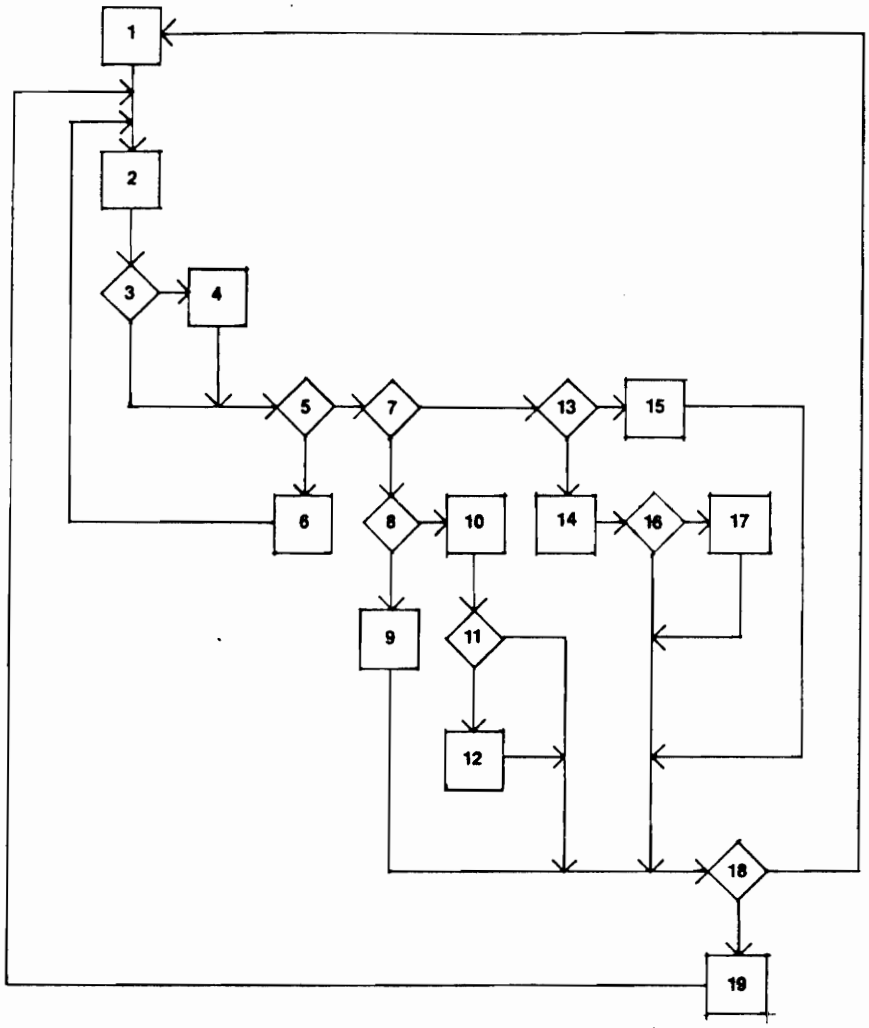
Algorithm 1.



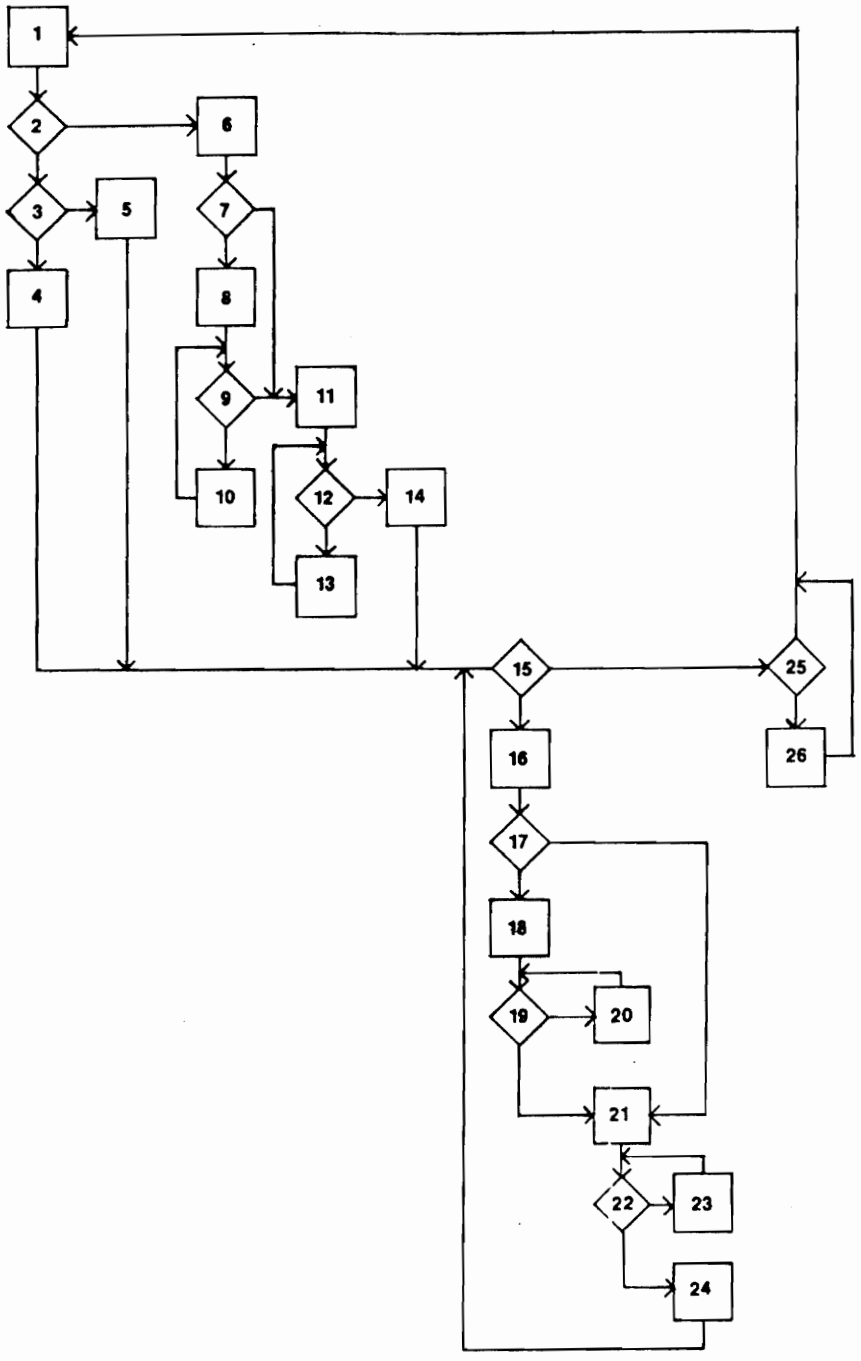
Algorithm 2.


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

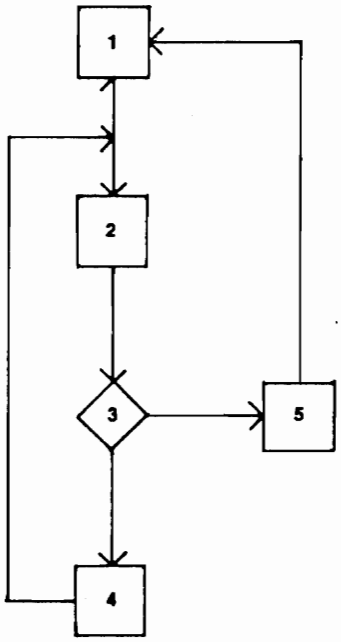
Algorithm 3



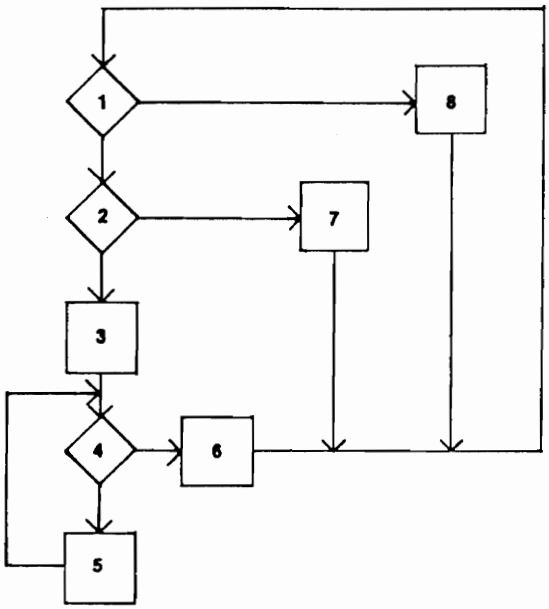
Algorithm 4.



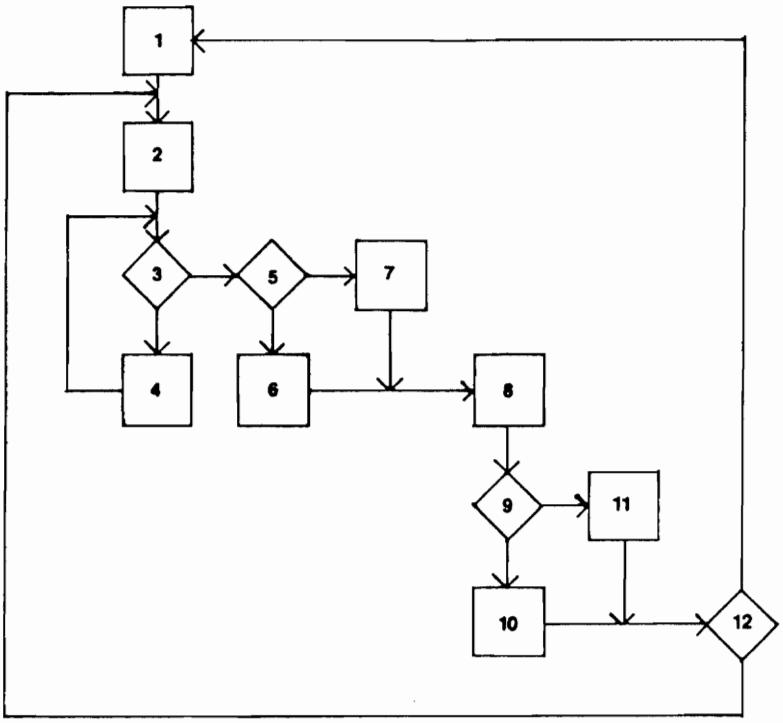
Algorithm 5



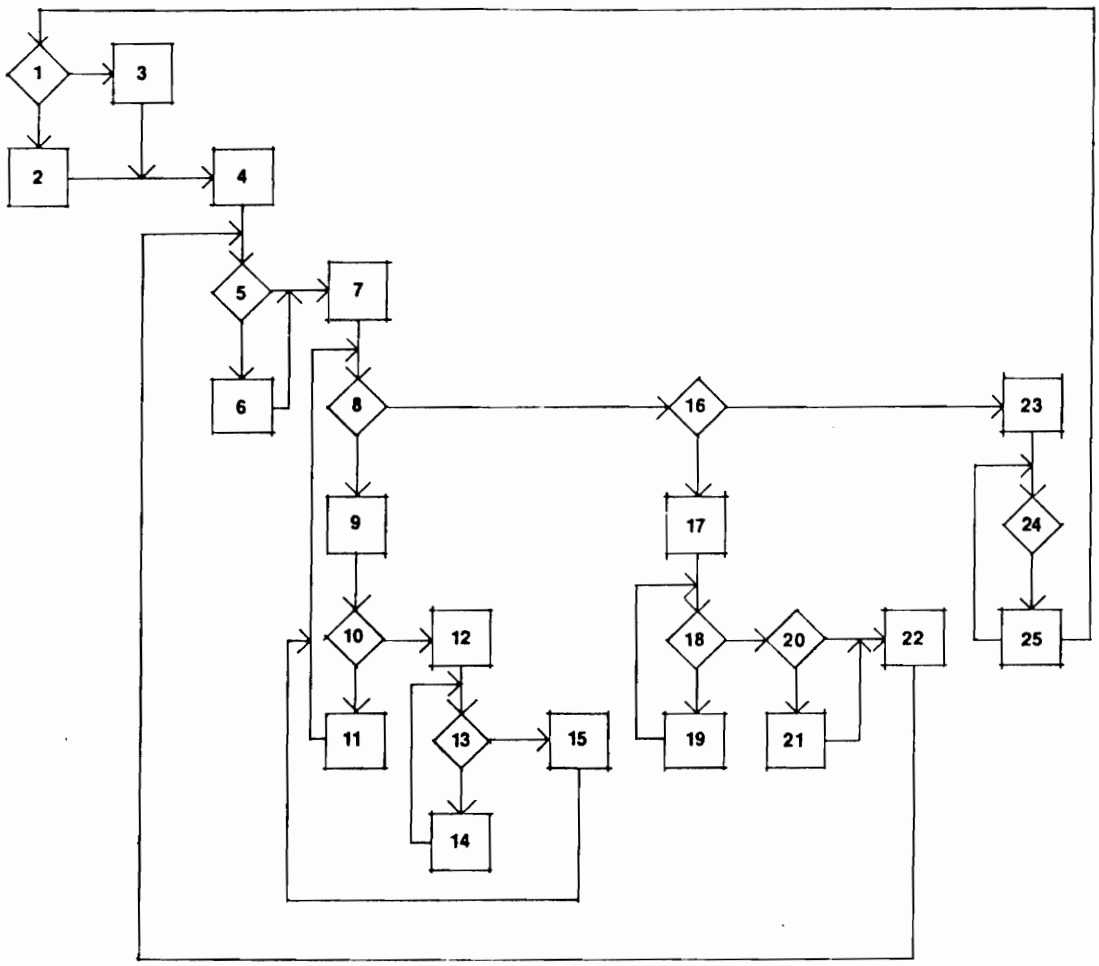
Algorithm 6.



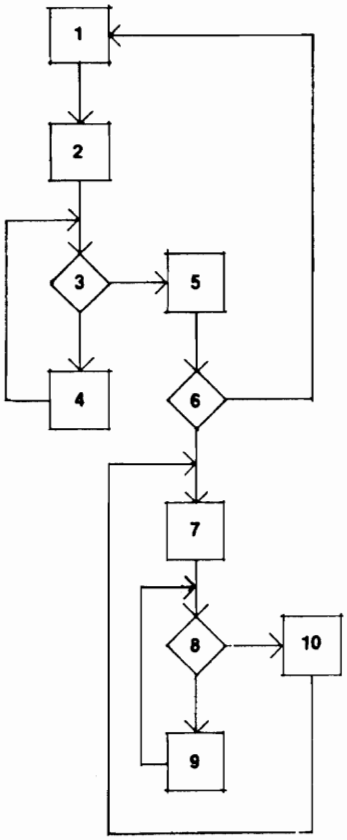
Algorithm 7.



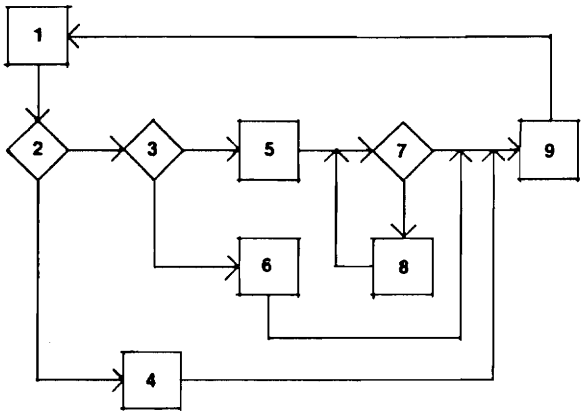
Algorithm 8.



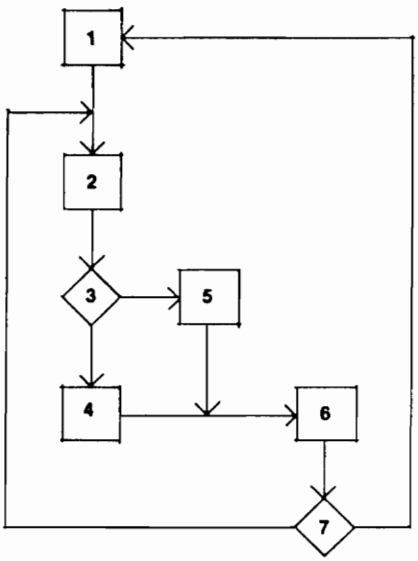
Algorithm 9.



Algorithm 9a.



Algorithms 10-13.

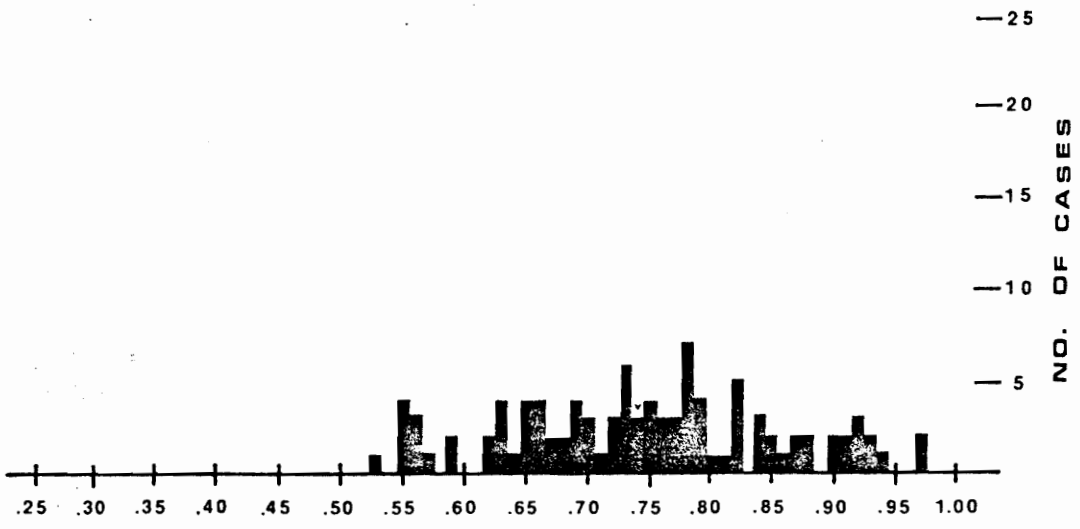


Algorithm 14.

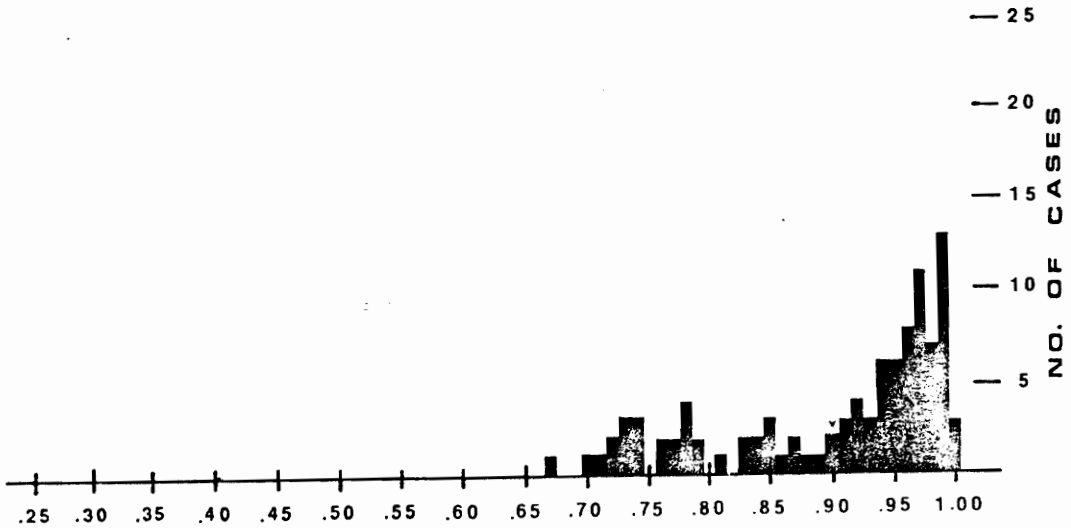
Appendix D.

Algorithms 1-14

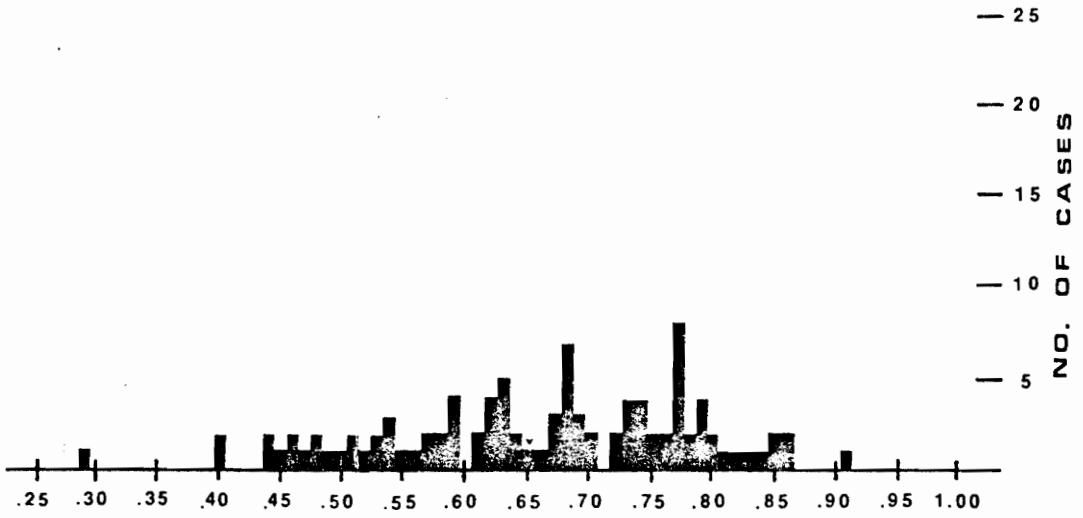
Distribution Histograms and Centroids from Model



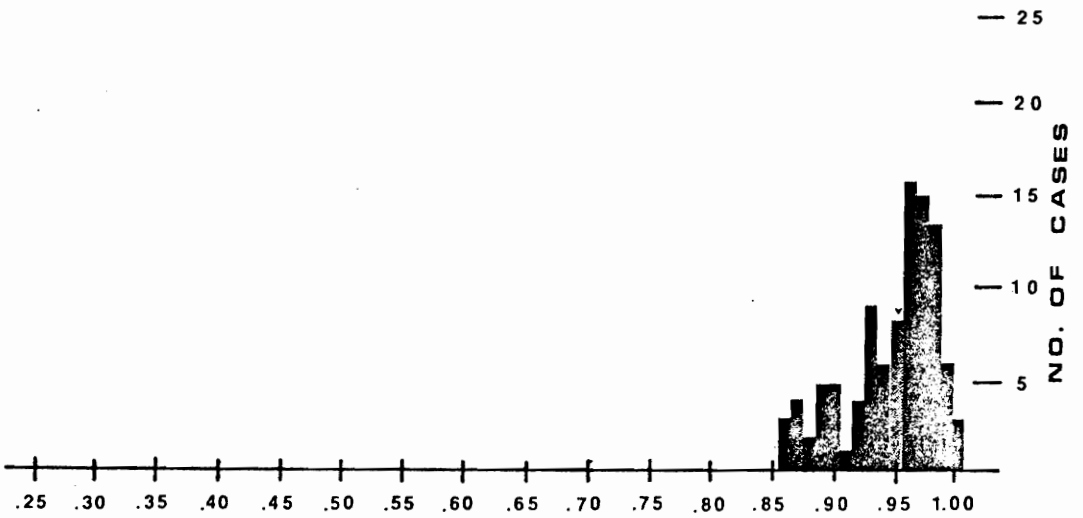
ALGORITHM 1 COSINES OF SAMPLES ABOUT CENTROID



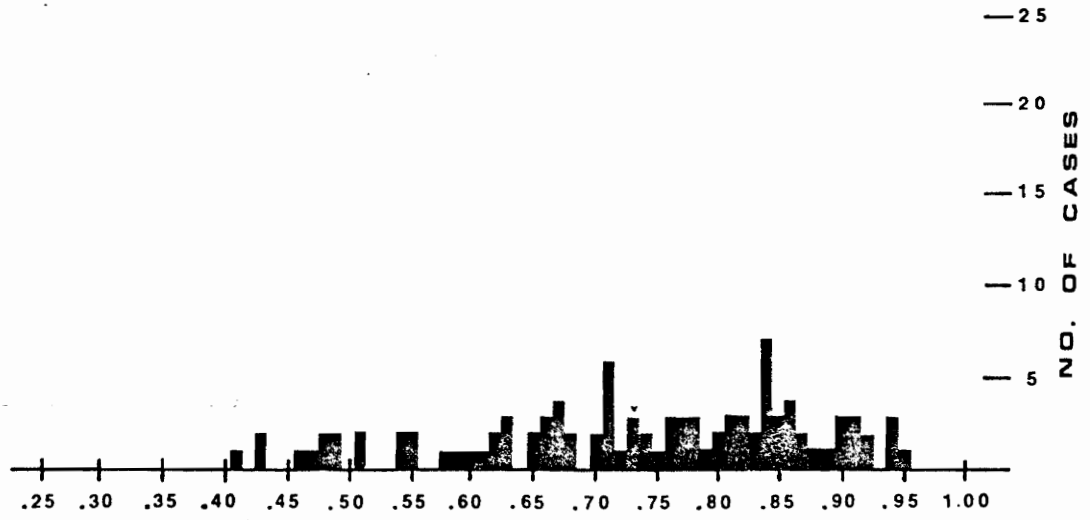
ALGORITHM 2 COSINES OF SAMPLES ABOUT CENTROID



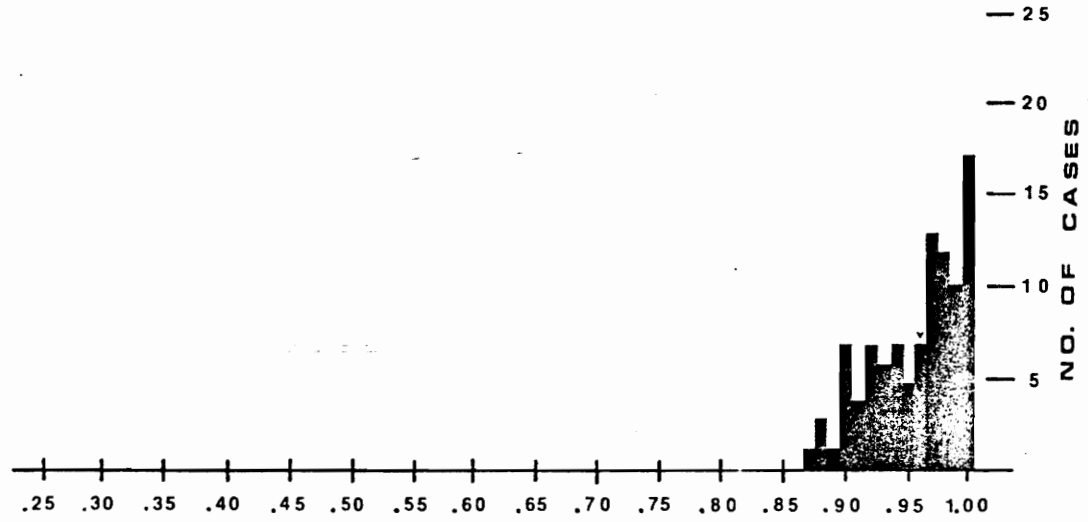
ALGORITHM 3 COSINES OF SAMPLES ABOUT CENTROID



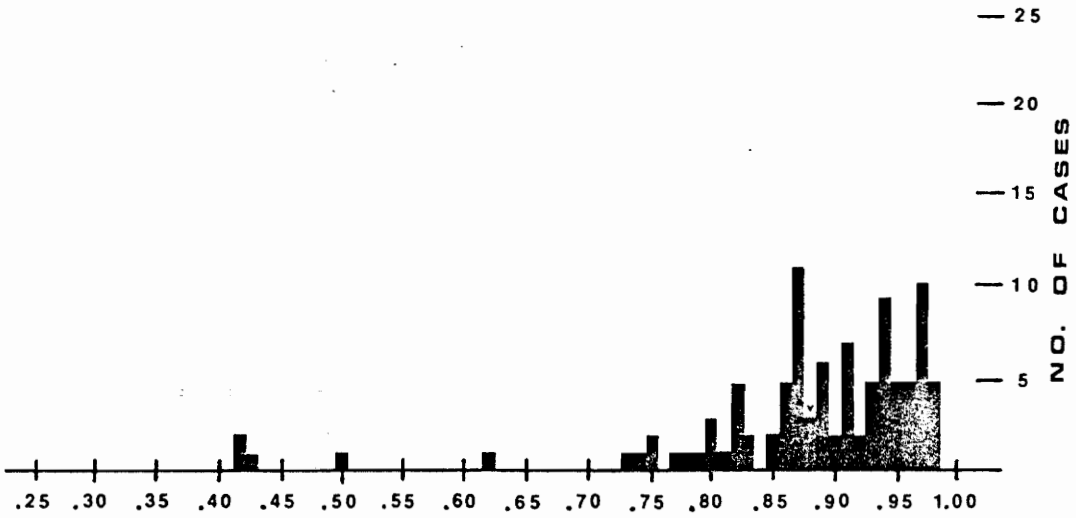
ALGORITHM 4 COSINES OF SAMPLES ABOUT CENTROID



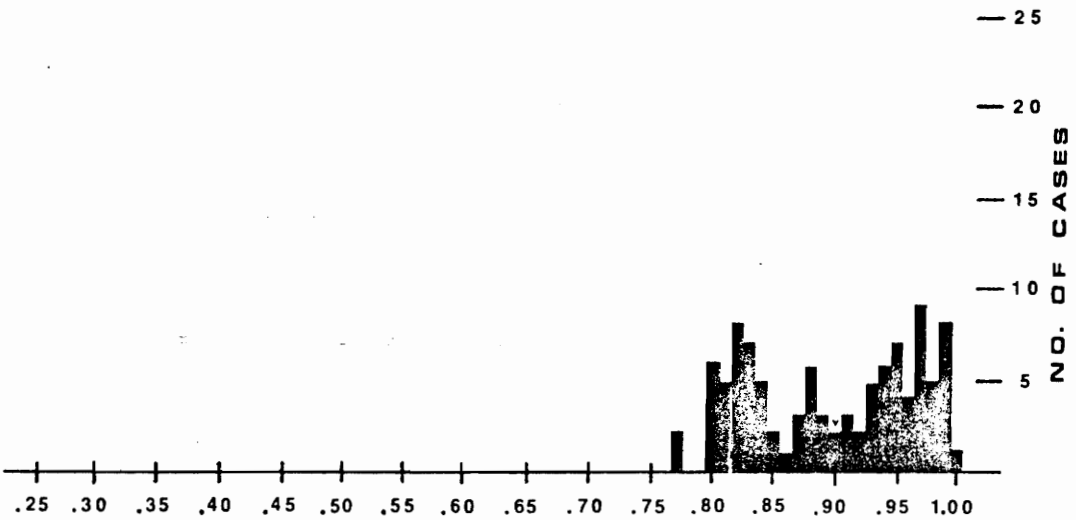
ALGORITHM 5 COSINES OF SAMPLES ABOUT CENTROID



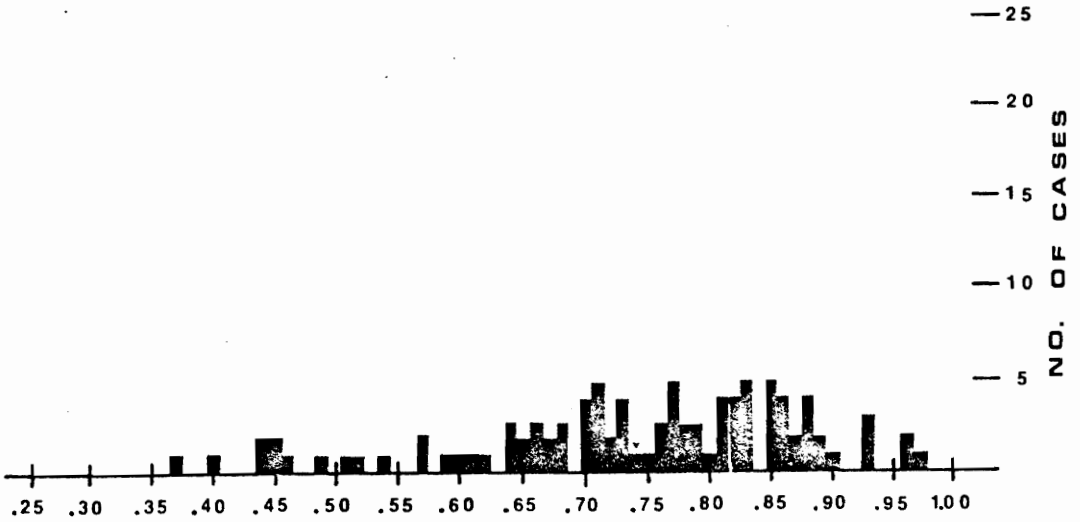
ALGORITHM 6 COSINES OF SAMPLES ABOUT CENTROID



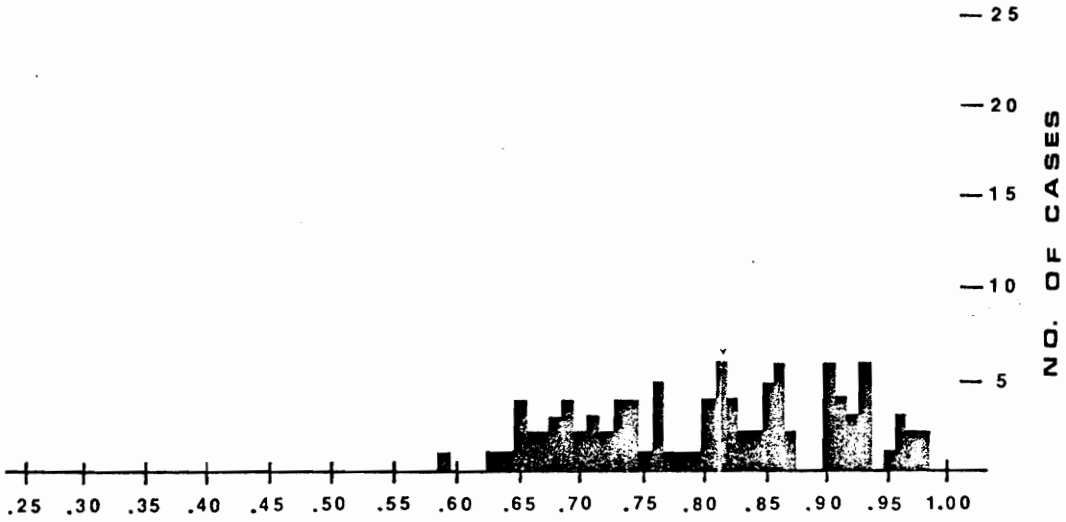
ALGORITHM 7 COSINES OF SAMPLES ABOUT CENTROID



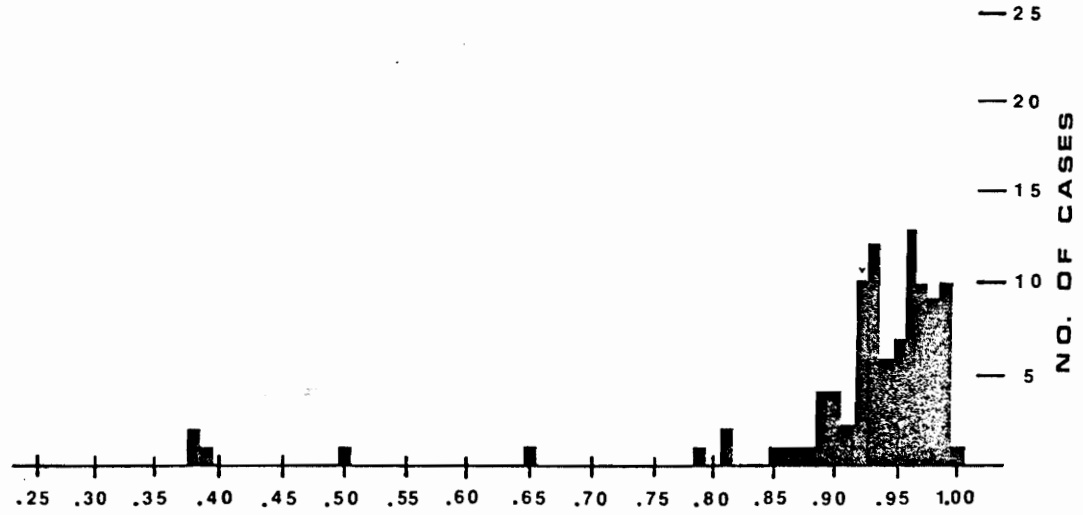
ALGORITHM 8 COSINES OF SAMPLES ABOUT CENTROID



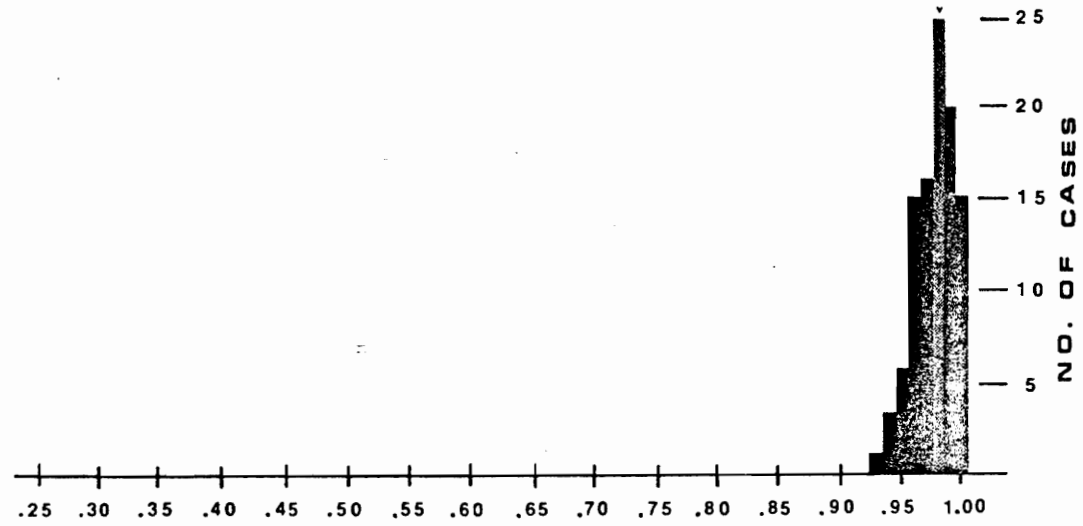
ALGORITHM 9 COSINES OF SAMPLES ABOUT CENTROID



ALGORITHM 9A COSINES OF SAMPLES ABOUT CENTROID



ALGORITHMS 10-13



ALGORITHM 14 COSINES OF SAMPLES ABOUT CENTROID

Algorithm 1.

.037 .113 .076 .037 .076 .039 .078 .039 .118 .079
.039 .039 .037 .089 .052 .052 .

Algorithm 2.

.213 .213 .085 .191 .191 .107 .

Algorithm 3.

.020 .055 .035 .020 .031 .010 .019 .022 .076 .055
.021 .046 .025 .087 .062 .025 .021 .021 .013 .009
.009 .004 .004 .002 .002 .001 .001 .010 .005 .005
.027 .027 .012 .015 .027 .014 .012 .036 .024 .012
.012 .007 .027 .016 .010 .005 .

Algorithm 4.

.037 .162 .162 .080 .162 .089 .034 .016 .008 .008
.008 .004 .018 .008 .010 .008 .004 .073 .037 .073

Algorithm 5.

.038 .038 .020 .011 .009 .019 .019 .009 .035 .026
.019 .063 .044 .019 .078 .039 .039 .020 .077 .057
.039 .112 .072 .039 .038 .020 .

Algorithm 6.

.142 .286 .286 .144 .142 .

Algorithm 7.

.298 .130 .058 .137 .078 .058 .072 .168 .

Algorithm 8.

.038 .085 .226 .140 .085 .040 .045 .085 .085 .040
.045 .085 .

Algorithm 9.

.025 .014 .011 .025 .046 .023 .046 .100 .054 .054
.029 .025 .085 .059 .025 .046 .021 .073 .052 .021
.012 .021 .025 .066 .041 .

Algorithm 9a.

.064 .064 .175 .111 .064 .124 .060 .169 .109 .060

Algorithms 10-13.

.188 .188 .088 .100 .041 .046 .101 .060 .188 .

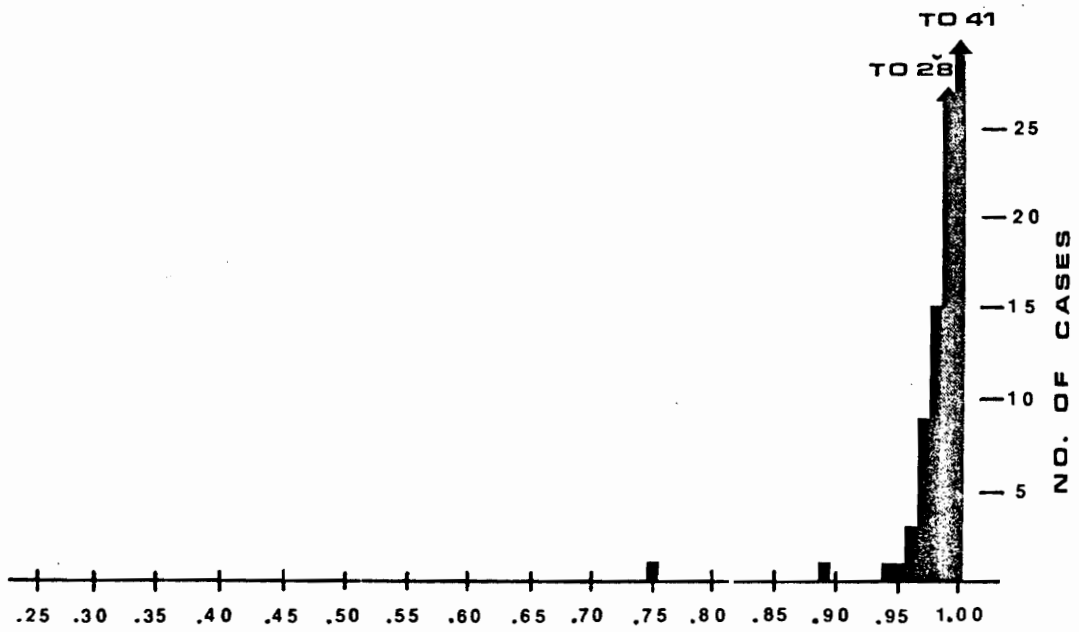
Algorithm 14.

.090 .182 .182 .092 .090 .182 .182 .

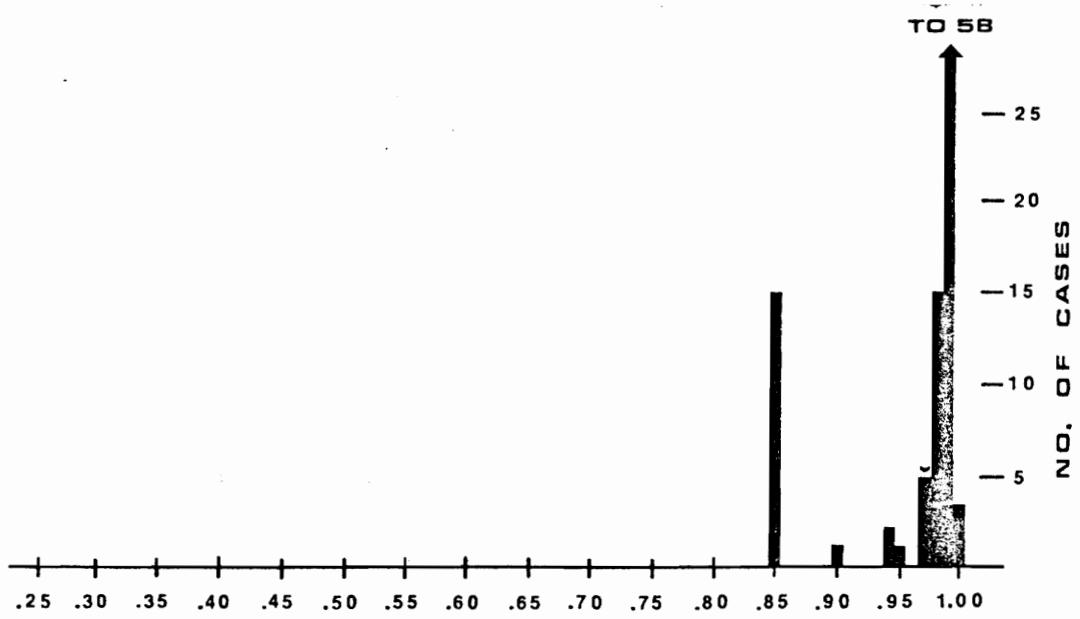
Appendix E.

Algorithms 3,4,5,7,10

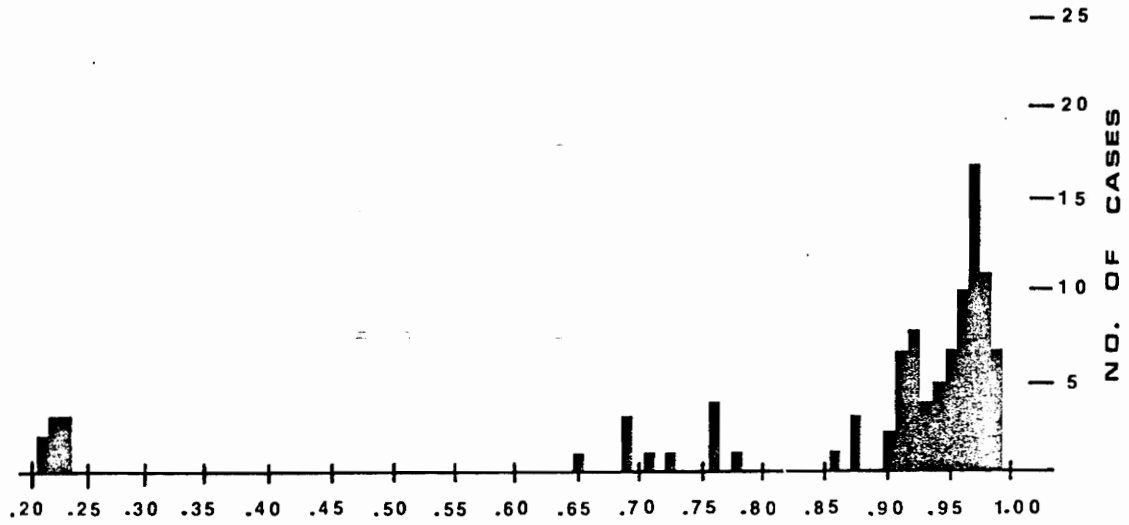
Distribution Histograms and Centroids from Runs



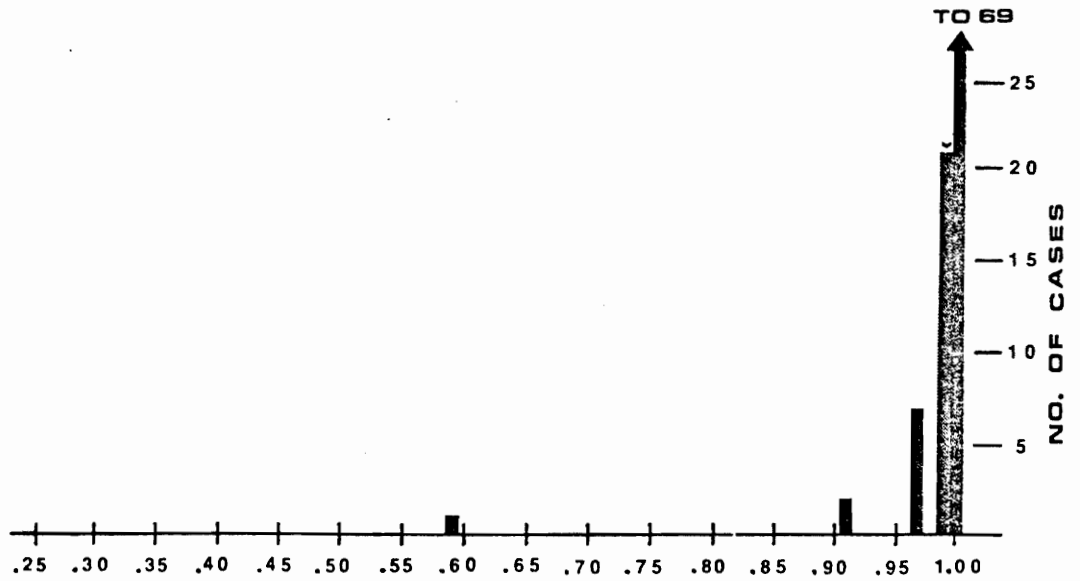
ALGORITHM 3(RUN) COSINES OF SAMPLES ABOUT CENTROID



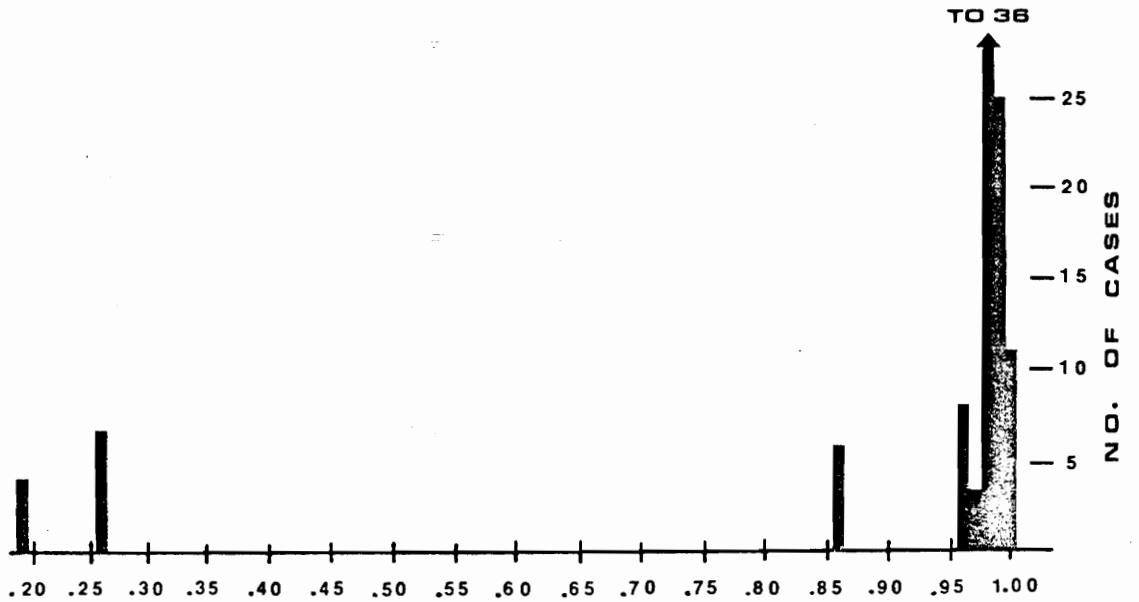
ALGORITHM 4(RUN) COSINES OF SAMPLES ABOUT CENTROID



ALGORITHM 5(RUN) COSINES OF SAMPLES ABOUT CENTROID



ALGORITHM 7(RUN) COSINES OF SAMPLES ABOUT CENTROID



ALGORITHM 10-13(RUN)

Algorithm 3.

.002 .013 .011 .002 .004 .003 .020 .017 .144 .127
.017 .052 .035 .185 .150 .035 .017 .017 .000 .017
.017 .000 .017 .001 .016 .001 .015 .003 .000 .002
.004 .004 .002 .002 .004 .000 .003 .014 .011 .003
.003 .001 .004 .001 .003 .001 .

Algorithm 4.

.018 .114 .114 .004 .114 .018 .091 .041 .007 .034
.034 .002 .050 .039 .011 .039 .002 .095 .077 .095

Algorithm 5.

.012 .012 .004 .004 .000 .008 .008 .000 .000 .000
.008 .025 .018 .008 .053 .041 .041 .041 .241 .201
.041 .106 .065 .041 .012 .012 .

Algorithm 7.

.075 .070 .070 .390 .320 .070 .000 .005 .

Algorithm 10.

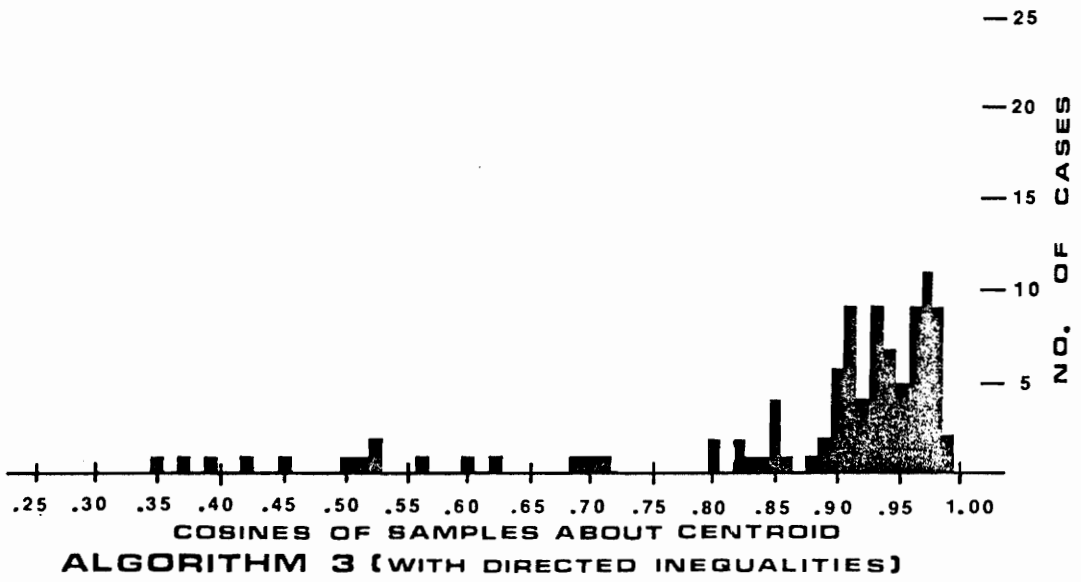
.076 .076 .056 .020 .050 .006 .355 .304 .056 .

Appendix F.

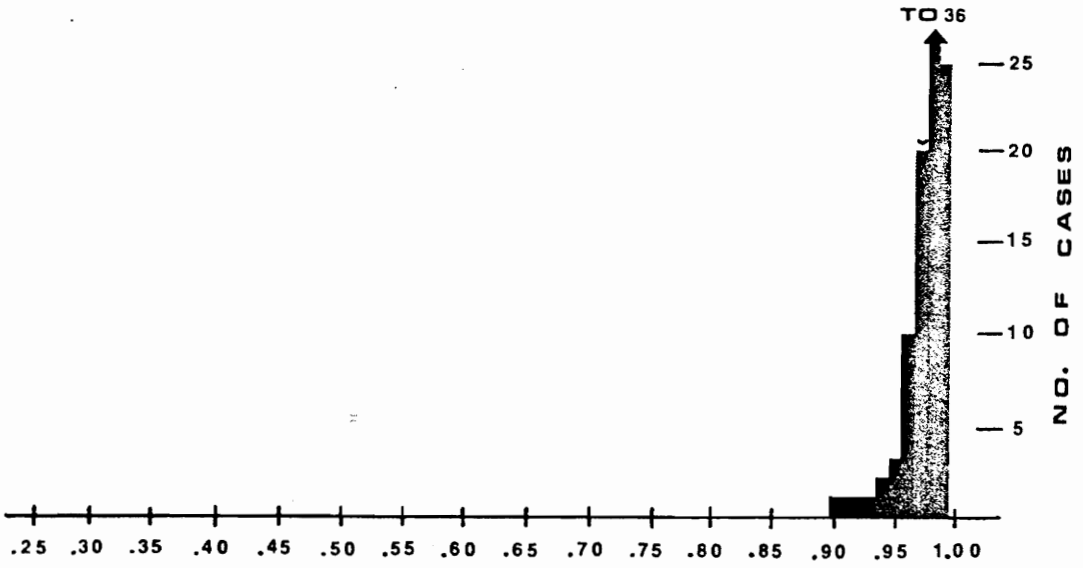
Algorithms 3,4,5,7,10

Distribution Histograms and Centroids

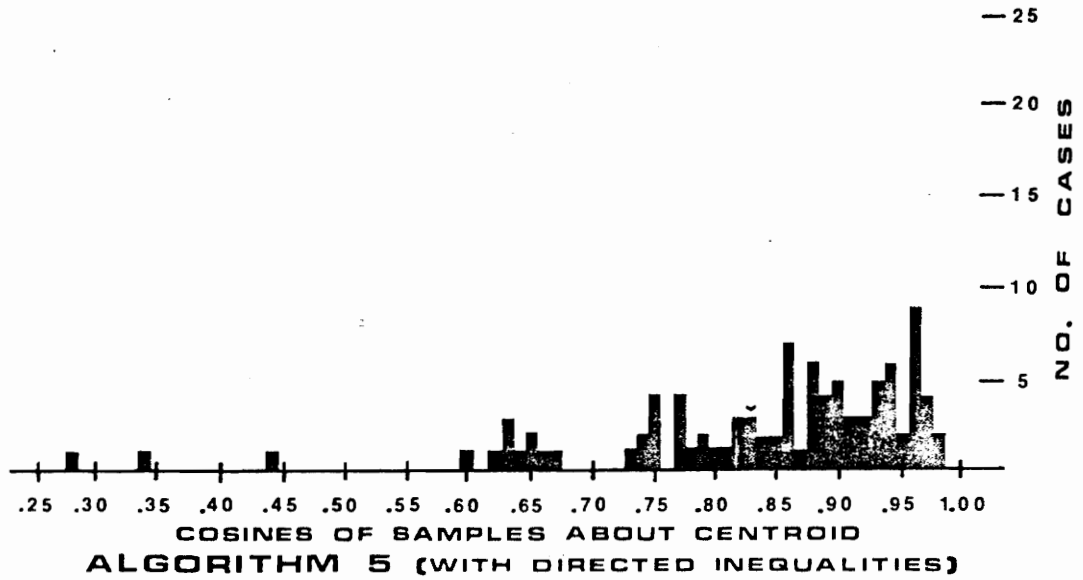
from the Modified Model



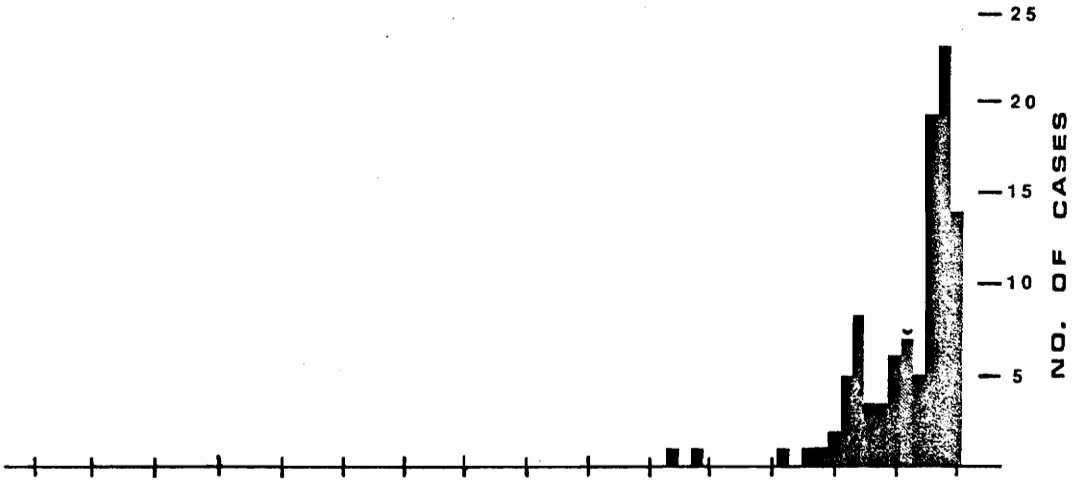
ALGORITHM 3 (WITH DIRECTED INEQUALITIES)



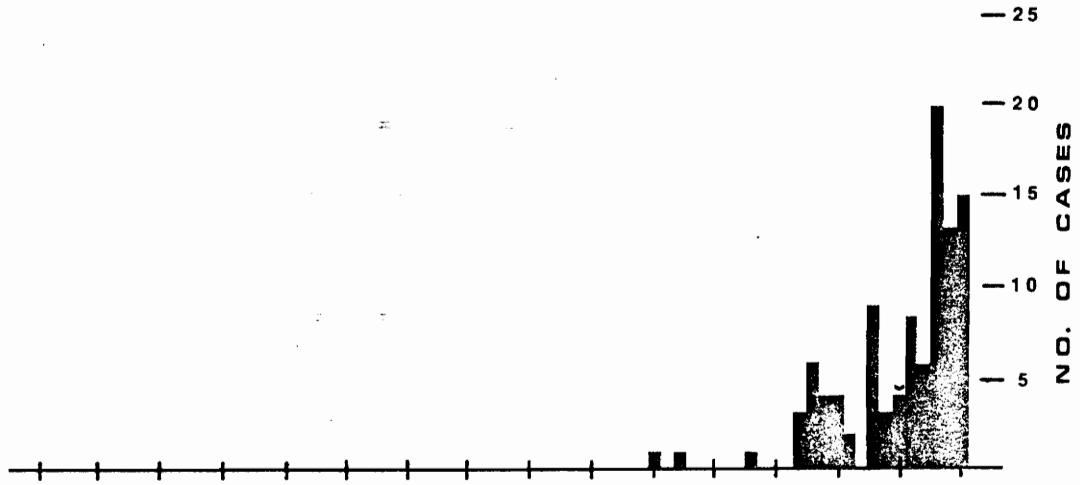
ALGORITHM 4 (WITH DIRECTED INEQUALITIES)



ALGORITHM 5 (WITH DIRECTED INEQUALITIES)



ALGORITHM 7 (WITH DIRECTED INEQUALITIES)



ALGORITHM 10 (WITH DIRECTED INEQUALITIES)

Algorithm 3.

.001 .009 .008 .001 .004 .003 .014 .015 .090 .075
.015 .067 .052 .248 .196 .052 .015 .015 .004 .011
.011 .003 .008 .002 .006 .001 .004 .003 .001 .002
.004 .004 .002 .002 .004 .003 .001 .014 .013 .001
.001 .000 .004 .001 .003 .001 .

Algorithm 4.

.024 .129 .129 .033 .129 .034 .070 .034 .018 .015
.015 .007 .037 .018 .018 .018 .009 .095 .071 .095

Algorithm 5.

.010 .010 .002 .002 .001 .007 .007 .002 .003 .001
.007 .065 .057 .007 .043 .033 .033 .026 .151 .125
.033 .178 .145 .033 .010 .007 .

Algorithm 7.

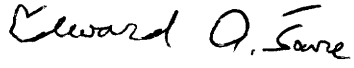
.127 .093 .067 .327 .259 .067 .025 .034 .

Algorithm 10.

.104 .104 .076 .027 .056 .020 .282 .226 .104 .

VITA

Edward A. Favre received his Bachelor of Science degree in Computer Science from Worcester Polytechnic Institute in June, 1975. He has been a graduate student in Computer Science and Applications at Virginia Polytechnic Institute and State University and will receive his Master of Science degree in that field in May, 1977.
Born March 15, 1953, he will shortly take a position as Software Engineer in the real world.



THE VALIDITY OF A MARKOV MODEL OF THE BEHAVIOUR OF PROGRAMS

by

Edward A. Favre

(ABSTRACT)

The thesis is concerned with the validation of a model of the runtime behaviour of a certain class of programs. It is shown that a Finite Markov Chain may be used to model the behaviour of a program that is built from single entry/single exit constructs, is global, and has one block in its flowgraph which possesses two cycles of relatively prime length. The sequence of powers of the program's transition matrix converges to a limit matrix, all of whose rows are identical. The column members of the limit matrix represent the relative frequency of execution of the blocks of the program. The validity of the model is tested by comparing modelling results with the run time statistics of Algorithms 1 through 14 of the Communications of the ACM. It is found that the original model does not predict the run time behaviour of some of the algorithms accurately. The model is modified by directing the inequalities of branch probabilities for decision node branches to DO loops and termination

statements. Finally, the algorithms are modelled by the modified process, resulting in predictions that are accurate enough for government work. [The work reported herein was sponsored in part by the National Science Foundation Grant No. DCR74-18108.]