Technical Report CS76004-R

A Markov Model of Certain Structured Programs

T. C. Wesselkamper

Richard W. Zoladz

Department of Computer Science

College of Arts and Sciences
Virginia Polytechnic Institute & State University
Blacksburg, Virginia 24061

ABSTRACT

The paper is concerned with modeling the run time behavior of a certain class of programs. Each program, represented by its flowgraph, is built up from one-in/one-out constructs (after the manner of Dijkstra). The programs have neither transient states nor absorbing states. Each program has one state which possesses two cycles of relatively prime length. A program which possesses these properties is called regular. Such a program may be modeled by a finite Markov chain. It is shown that if a program is regular then its Markov model has a regular transition matrix, that is, the sequence of powers of the transition matrix converges to a matrix all of whose rows are identical.

The experimental validity of the method is discussed, as are the implications of the method for program design.

C. R. Categories: 8.1, 5.32, 4.20

AMS (MOS) : 60 J 10, 05C20

A MARKOV MODEL OF CERTAIN STRUCTURED PROGRAMS

by

T.C. Wesselkamper

and

Richard W. Zoladz

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24060

ABSTRACT

The paper is concerned with modelling the run time behavior of a certain class of programs. Each program, represented by its flowgraph, is built up from one-in/one-out constructs (after the manner of Dijkstra). The programs have neither transient states nor absorbing states. Each program has one state which possesses two cycles of relatively prime length. A program which possesses these properties is called regular. Such a program may be modelled by a finite Markov chain. It is shown that if a program is regular then its Markov model has a regular transition matrix, that is, the sequence of powers of the transition matrix converges to a matrix all of whose rows are identical.

The experimental validity of the method is discussed, as are the implications of the method for program design.

C.R. Categories : 8.1, 5.32, 4.20
AMS (MOS) : 60J10, 05C20

Technical Report CS 76004-R

I. Preliminaries.

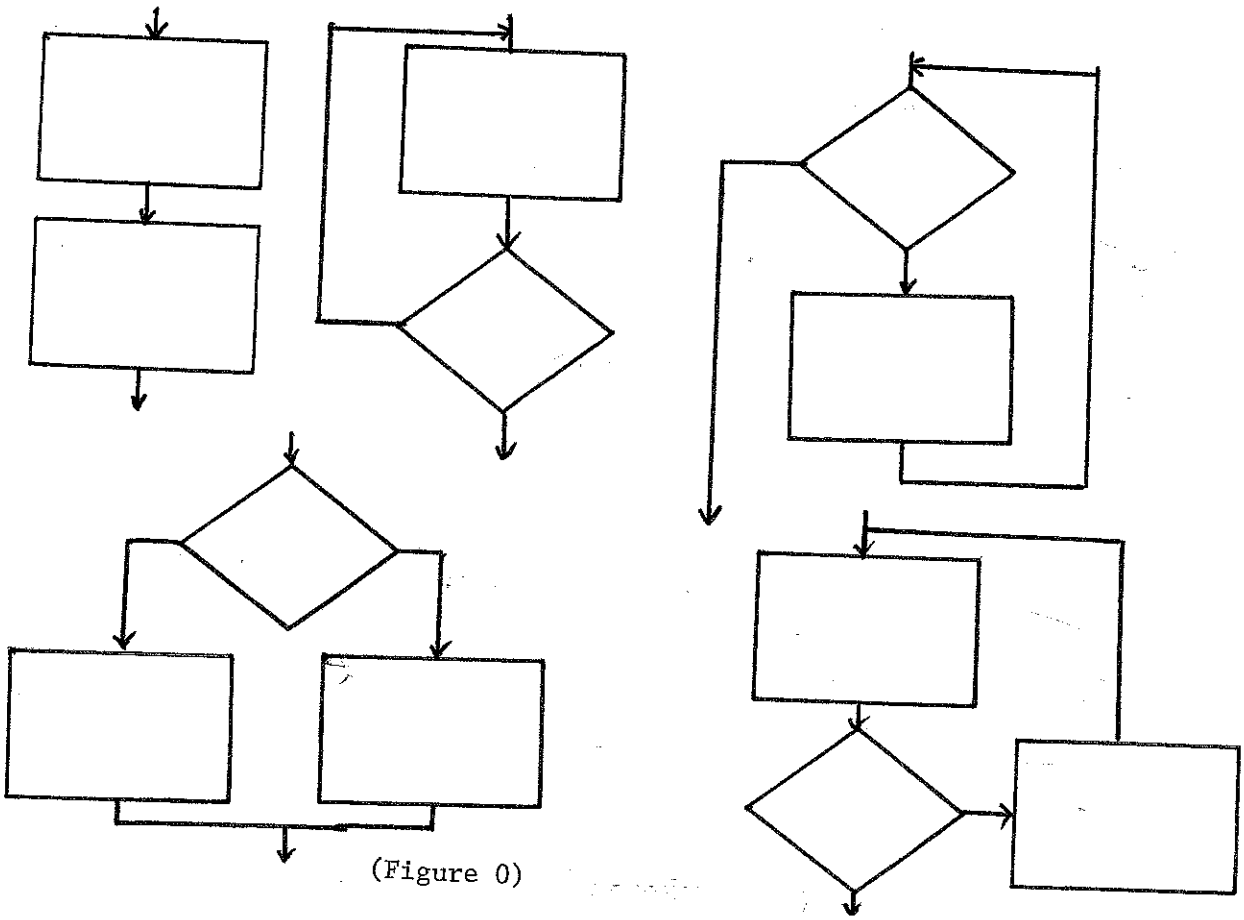
In recent years computer vendors have made available user microprogrammable processors. This has made possible a strategy of optimizing the execution time of programs by dynamically modifying the virtual machine upon which the program executes so as to produce the fastest possible execution. In particular, the following has been suggested: Suppose that the code for a program exists in source form. The code is to be compiled and resulting object code executed. Firstly, configure the host computer to a configuration which is optimal for handling the task of compilation. Compilation complete, analyze the produced object code and on the basis of this analysis configure the machine which will execute this code most efficiently. The accomplishment of this task depends upon the ability of the reconfiguration system to choose those instructions in the object code which execute very frequently and produce a virtual machine which executes the instructions very quickly. In practice this has come to mean implementing the very frequently executed instructions in microcode.

In the light of the last fact we may reformulate the goal of the exercise: Suppose the code for a program exists in source form. Analyze this source code to determine which source statement level operations will be executed most frequently. In the process of compilation compile these most frequently executed instructions into microcode and compile less frequently executing

instructions into the code of the host virtual machine. In other words, enhance the instruction set of the host virtual machine by adding to that instruction set additional instructions which implement the most frequently executed source statement level instructions.

It is to this reformulated task we address ourselves. We take a "program" to be a process represented by a flowgraph. We must now by a sequence of restrictions limit this even farther, for there are programs whose behavior we do not attempt to analyze. Our goal is to provide a means of determining what are these most frequently executed instructions.

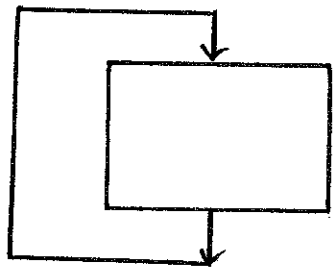
We limit ourselves firstly to flowgraphs which are "structured". Specifically, we limit ourselves to flowgraphs which are recursively developed by the replacement of an action block (rectangle) by one of the following five constructs:



(Figure 0)

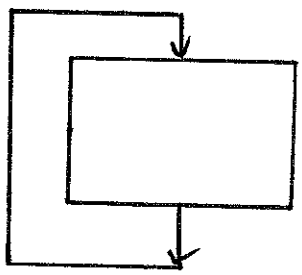
The rationale for limiting ourselves to these constructs is in Martin [1].

Secondly, we limit ourselves to flowgraphs which are nonterminating (do not have a "halt" state) and do not have transient states. Specifically, we assume that at its most global level every flowgraph has the form:



This implies not only that the process modelled by the flowgraph is nonterminating but also that there exists no set of states which, once left, is never reentered.

In a formal way we may describe this set of flowgraphs as a language with goal symbol:



(Figure I)

and with the five productions of the form:

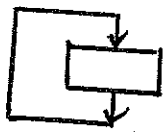
where  is one of the constructs of Figure 0.

(Figure II)

Thirdly, within this set of flowgraphs we select a subset which we call the subset of <u>regular flowgraphs</u>. We note that because of the two above limitations each decision block and each action block of a flowgraph has the property that there exists at least one path which begins at a state and leads back to that state. We call such a path the <u>cycle</u> of the state and define the length of the cycle as the number of intermediate edges in the cycle. If a cycle is of length n we call it an n-cycle. The subset of regular flowgraphs is characterized by the property that it contains (at least) one state which possesses a p-cycle and a q-cycle where p and q are relatively prime integers, that is, $(p,q) = 1$.

Definition: A flowgraph is regular if it satisfies the following:

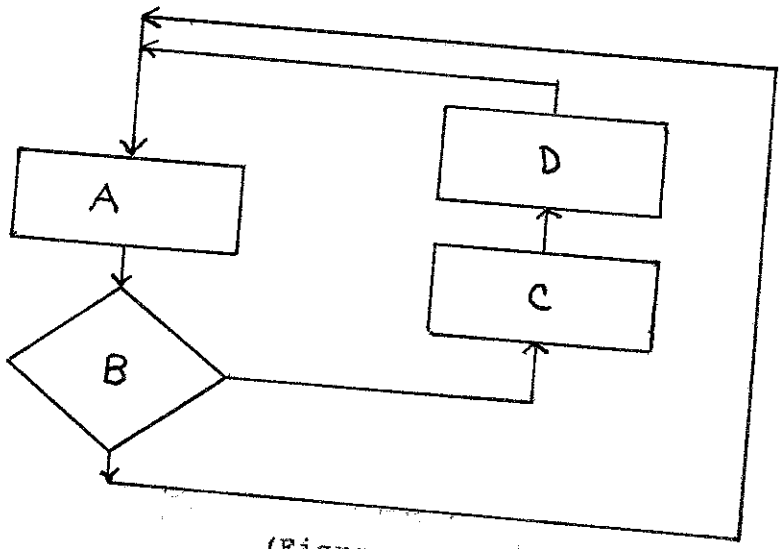
1) The flowgraph has as its goal symbol



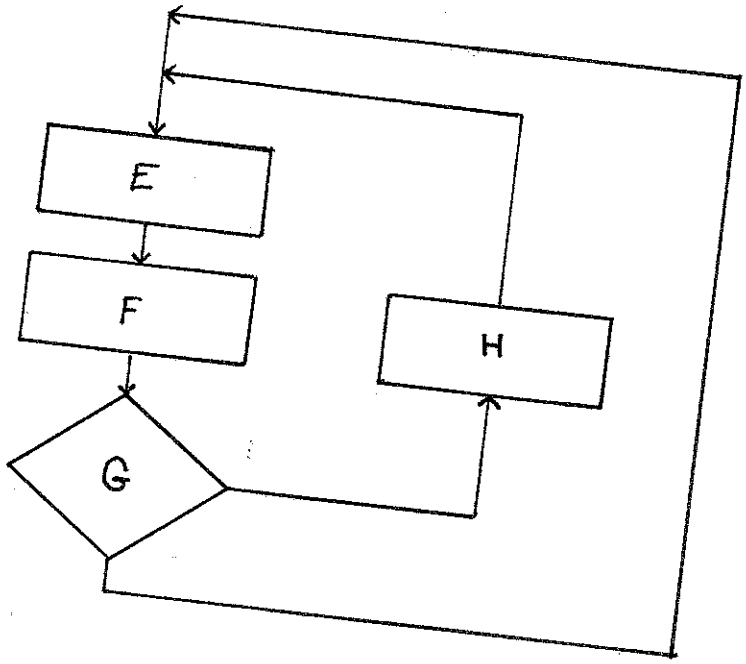
2) The flowgraph is developed from 1) by the recursive application of the substitutions of Figure II.

3) The flowgraph has a state which possesses two cycles of relatively prime length.

Thus: The flowgraphs III and IV

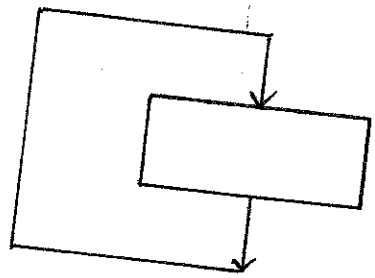


(Figure III)

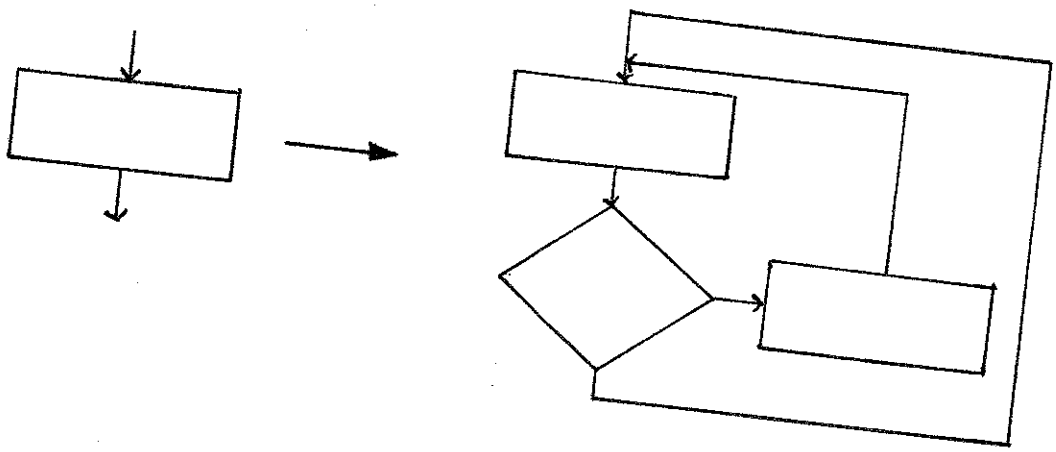


(Figure IV)

are both derived from



by the application of the two transformations:



and

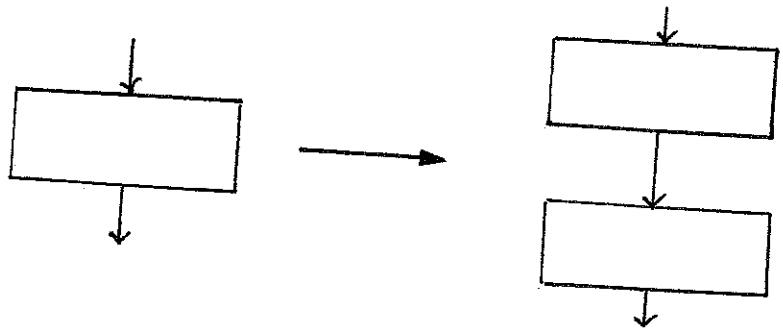


Figure III is not a regular flowgraph since each cycle of each state is of even length; Figure IV is regular since state E has a cycle of length 3 (EFGE) and a cycle of length 4 (EFGHE) and (3,4) = 1.

Finally we define a directed distance between two states of a regular flowgraph:

Definition: If P and Q are two distinct states of a regular flowgraph, let d(P,P) = 0 and d(P,Q) denote the length of the minimal path from P to Q.

Note that this directed distance function is positive definite and satisfies the triangle inequality, but is not symmetric.

II. MARKOV PROCESSES.

A Markov process is a process with a finite number of states, denoted {a1, a2, ..., an}, such that at most one state change is possible during a single event. In addition, the probability that the process will be in any given state can depend at most upon the immediately previous state of the process, and this probability must be known. The last assumption essentially requires that the probability of an event moving the process from state ai to state aj remains fixed through time - it is entirely independent of the previous history of the process.

The probability that the process will move from state ai to state aj during one event is denoted p(i,j), or simply pij. Such probabilities are called transition probabilities, and the matrix

$$P = \begin{bmatrix} p11 & p12 & p13 & \dots & p1n \\ p21 & p22 & p23 & \dots & p2n \\ \dots & \dots & \dots & \dots & \dots \\ pn1 & pn2 & pn3 & \dots & pnn \end{bmatrix}$$

is called the transition matrix of the process. Since during one event the process must either stay in the same state or move to

one of the finite number of other possible states and since each element of P must be non-negative, the elements of each row must sum to one. Each row of P is called a probability vector and P is called a stochastic matrix.

P describes the possible effects of a single event of the process. It is also useful to be able to determine the probability that the process which began in state ai will be in state aj after m events, for an arbitrary integer m. This probability is denoted by $p_{ij}(m)$. Consider $p_{ij}(2)$, the probability that a process which began in state ai will, after two events, be in state aj. For each k, $1 \leq k \leq n$, the process could move first from ai to ak and then from ak to aj; hence $p_{ij}(2) = p_{i1}p_{1j} + p_{i2}p_{2j} + \ldots + p_{in}p_{nj}$. This is precisely the entry in the i-th row and the j-th column of P^2. In general,

$$P(m) = \begin{bmatrix} p_{11}(m) & p_{12}(m) & \ldots & p_{1n}(m) \\ p_{21}(m) & p_{22}(m) & \ldots & p_{2n}(m) \\ \ldots & \ldots & \ldots & \ldots \\ p_{n1}(m) & p_{n2}(m) & \ldots & p_{nn}(m) \end{bmatrix}$$

P(m) is called the m-th order transition matrix and contains the probabilities of the state of the process after m events.

A stochastic matrix is called regular if some power of the matrix has only positive entries. A set of states is called ergodic if it is possible to go from every state in the set to every other state in the set during a finite number of events. Thus, all regular chains are ergodic, since if P(n) is the power of P with all positive entries, it is possible to move from every state to every other state during n or fewer events.

Regular Markov Chains are exceptionally well-behaved as can be seen from the following theorem (an analogous result also exists for ergodic chains in general):

If P is a regular stochastic matrix, then

(1) The powers P(n) approach a matrix T (called the equilibrium matrix);

(2) Each row of T is the same probability vector t = (t1,t2, ... ,tn);

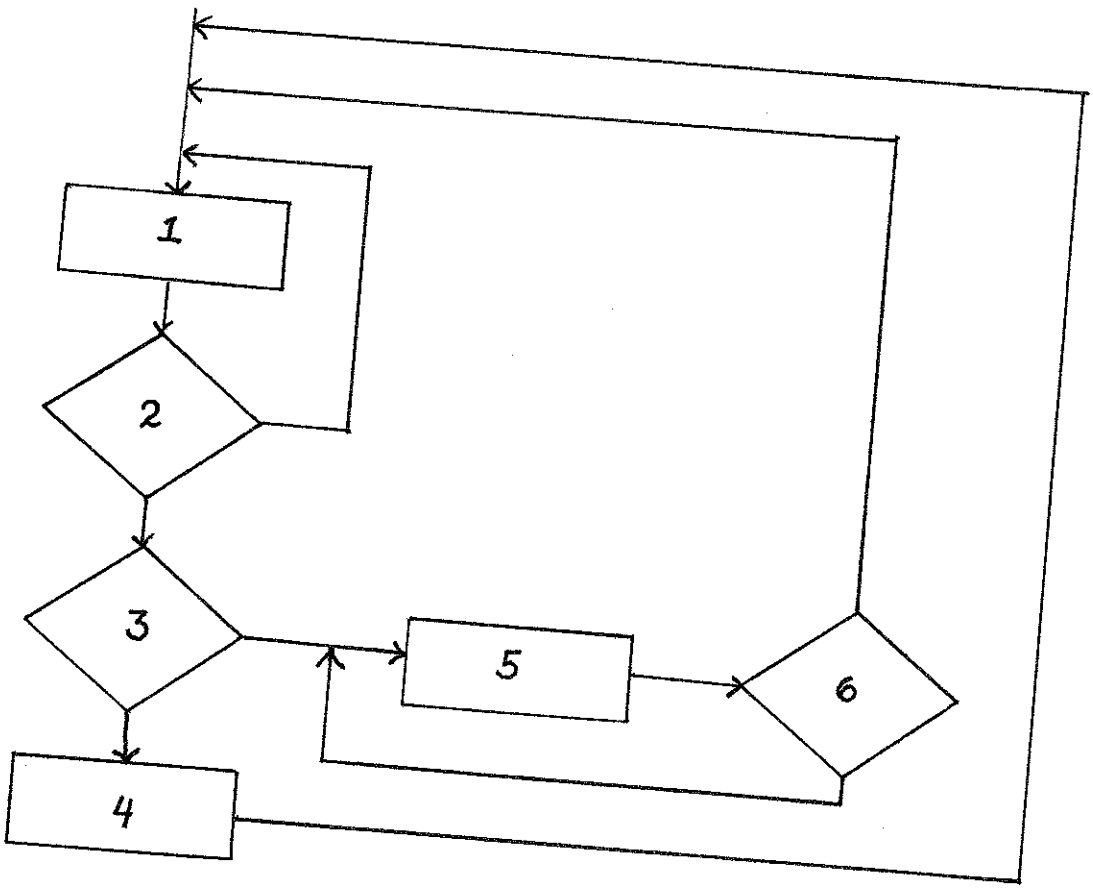
(3) The components of t are positive;

$$P(n) \rightarrow T = \begin{bmatrix} t1 & t2 & \dots & tn \\ t1 & t2 & \dots & tn \\ \dots & \dots & \dots & \dots \\ t1 & t2 & \dots & tn \end{bmatrix} ; \; ti > 0$$

This implies that after a sufficiently large number of events, the probability that the process will be in state aj will be

nearly tj, regardless of the starting state of the process. In other words, long range predictions can be made independently of the starting state. It is in this property of regular Markov Chains that we are primarily interested.

In this paper we propose to examine the flowgraph of the source level program and to model the source level states of the flowgraph as states in a Markov process. Consider the simple scanning routine recently used by Wirth as an example of a structured program [2].



(Figure V)

This program receives as input linear text which contains blank characters and comments in addition to other text. Comments have the form: {<comment>}. The above flowgraph represents a program to delete blanks and comments from the linear text and to output the resulting edited text. When the program is in a state represented by an action block (a rectangle) there is a unique "next state" to which the program moves with probability 1. When the program is in a state represented by a decision block (a rhombus) there exist non-zero probabilities p and q, (p+q=1), and two states, say A and B, such that the next state of the program will be A with probability p and will be B with probability q. Thus the state transition matrix for the above flow graph has the form:

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

where,

$$p + q = 1;$$

$$r + s = 1;$$

$$t + u = 1.$$

III. THE MAIN RESULT.

In the proof of the main results of this paper we use a lemma from elementary number theory.

Lemma 1: If m and n are positive integers such that $(m,n) = 1$, then the Diophantine equation:

$$mx + ny = c$$

has solutions in positive integers, x and y, for all c such that $mn - m - n < c$. (See for example [3].)

Now let us suppose that G is a regular flowgraph and suppose that S is a state which possesses two cycles of relatively prime length, say m and n.

Lemma 2: There exists a positive integer c such that for each $p > c$, the state S possesses a cycle of length p.

proof: Let $c = mn - m - n$. By Lemma 1 the Diophantine equation $mx + ny = P$ has a solution in positive integers since $P > c$. Say that the solution is $(x0, y0)$. Beginning at S, traverse the m-cycle x0 times and then traverse the n-cycle y0 times. The resulting cycle is of length $mx0 + ny0 = P$.

Lemma 3: Each state T of a regular flowgraph G possesses two cycles of relatively prime length.

proof: The regular flowgraph G has a state S which possess two cycles of relatively prime length. By Lemma 2 there exists a positive integer $c(S)$ such that S possesses a cycle of length c' for each $c' \geq c(S)$. Let $s' = d(S,T)$, the directed distance from S to T, and let $t' = d(T,S)$, the directed distance from T to S. Let p' and q' be two relatively prime integers, each greater than

$s' + t' + c(S)$. Now note that $p' - s' - t' > c(S)$ and so S possesses a cycle of length $p' - s' - t'$. Hence the path from T to S followed by the cycle of length $p' - s' - t'$ around S, followed by the path from S to T is a cycle of length p' belonging to T. Similarly there is a cycle of length q' belonging to T.

Corollary: If G is a regular flowgraph then with each state S of G there is associated a minimal constant $c(S)$ such that for each integer $c' \geq c(S)$, S possesses a cycle of length c'.

Lemma 4: If S and T are two states of a regular flowgraph G, then there exists a constant $c(S,T)$ such that for each integer $c' \geq c(S,T)$, there exists a path of length c' from S to T.

proof: Let $c(S,T) = c(S) + d(S,T)$. For each $c' \geq c(S,T)$, cycle around S, $c' - d(S,T)$ times, then go to T.

Hereafter, we let $c(S,T)$ denote the minimum such path.

Theorem: Each regular flowgraph G is modeled by a Markov process with a regular transition matrix T.

proof: We need to prove that there exists an integer n such that $T(n)$, the n-th power of T, has no non-zero entries. Let $n = \max \{c(A,B) \mid A,B \text{ are states of } G\}$. If P and Q are two states of G then $n \geq c(P,Q)$ and so there is a path of length n from P to Q. Hence the entry in the P-th row and Q-th column of $T(n)$ is non-zero.

IV. AN EXAMPLE.

Returning to the example in Figure V, note that state 1 has a cycle of length 2 (121) and a cycle of length 5 (123561). Hence for each c > 2 * 5 - 5 - 2 = 3, there is a cycle of length c. Each other state also has two cycles of relatively prime length, namely:

state 2: (212) and (235612)

state 3: (34123) and (356123)

state 4: (41234) and (4123561234)

state 5: (565) and (561235)

state 6: (656) and (612356).

The largest of the integers, c(S), is c(3) = 23. Since the longest distance between any two states is 5 (412356) it follows that P(28) contains all non-zero entries.

Suppose further that P is the matrix

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ .5 & 0 & .5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .98 & .02 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ .02 & 0 & 0 & 0 & .98 & 0 \end{bmatrix}$$

(Figure VI).

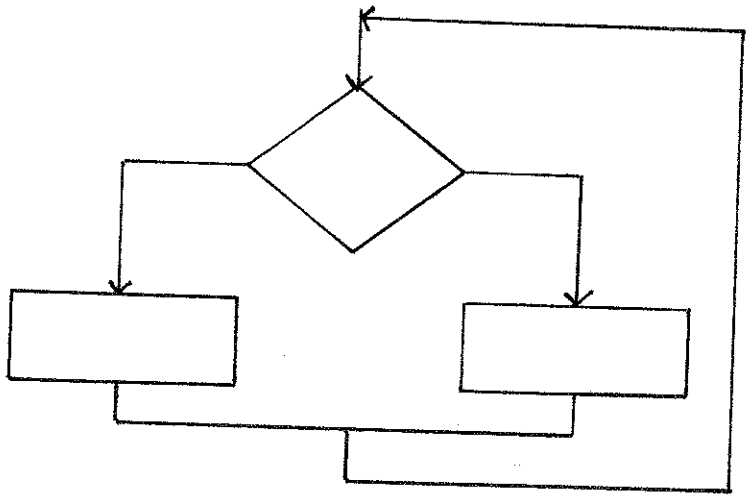
Then the limiting vector T is:

$$T = (.25 \quad .25 \quad .12 \quad .12 \quad .13 \quad .13).$$

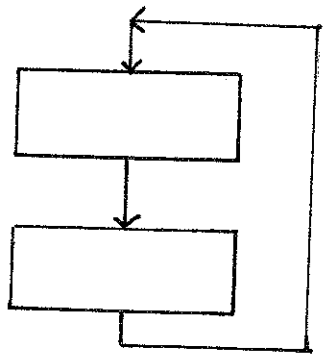
V. SUMMARY.

The effect of the model is to give a method of approximating from source code the run time behavior of a program. As such it has been successfully used by T. G. Rauscher in his work with the dynamic redefinition of architecture [4]. Regrettably his statement of the main theorem of his paper is incorrect as is his proof. He does not observe the necessity of the condition that some state have two relatively prime cycles [4, p. 61].

In this paper we have shown the sufficiency of this condition of the existence of relatively prime cycles. A proof of necessity is straight forward. Without such a condition a program such as



may occur, which is ergodic with period 2, or even more simply



However, in real programs, the condition seems to take care of itself. The writers suspect that a better formulation of the notion of structured programs might cure the defect.

There is, however, a deeper problem. It might be formulated as follows: Now see here, a program is simply not a Markov process. Given a state, the probabilities of which states will be visited next are data dependent (although of course which states have zero probabilities and which states have nonzero probabilities are not).

This leads to the following experiment. What happens if one perturbs the values of the nonzero probabilities in a row of the transition matrix, leaving the sense of the inequality between the nonzero probabilities unchanged. The answer appears to be: nearly nothing happens. For example, if the transition matrix of Figure VI is replaced by:

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

thereby perturbing the third and sixth rows, the limiting vector is:

(.26 .26 .13 .09 .13 .13).

This leads the authors to suspect that there is a deeper set

of results than those presented in this paper.

VI. REFERENCES.

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

2. Niklaus Wirth, On the Composition of Well-Structured Programs, ACM Computing Surveys 6, no. 4 (December 1974), pp. 247-259.

3. Solution to Problem E1392, American Mathematical Monthly 67, no. 6 (June-July 1960), p. 594.

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