Models of Computations and Systems—Evaluation of Vertex Probabilities in Graph Models of Computations

DAVID MARTIN AND GERALD ESTRIN

University of California at Los Angeles, Los Angeles, California*

ABSTRACT. This paper concerns itself with the modeling of computations and systems and the generation of a priori estimates of expected computation time for given problems on given processing systems. In particular, methods are discussed for determining the probabilities of reaching vertices in a graph model of computations.

A priori estimates of expected computation time for given problems on given processing systems may be generated by modeling the computation with a transitive directed graph [1].

A computational algorithm is first represented by a directed graph containing cycles, with vertices representing macro-operations and arcs representing sequence, branching control conditions and data transfer. Cycles may then be removed in a systematic transformation resulting in a transitive directed graph [2–5]. The model of the computation can be assigned to a model of a computer system and a successive relaxation procedure used to obtain a suboptimal assignment and sequencing of tasks on machines. In this process a measure of expected path length, which takes into account branching probabilities, serves as a criterion.

A fundamental aspect of the above process involves the computation of vertex probabilities which are then used in computing the estimates of expected path length through a graph.

In this paper the nature of transitive directed graphs representing computations is discussed, a systematic enumerative procedure for calculating vertex probabilities is described, and then a more practicable algorithm useful in a priori assignment and sequencing experiments is established.

A computer program for calculating vertex probabilities is presented and results summarized for several graphs abstracted from complex problems.

Graph Model of Computation

Consider a graph (such as Figure 8) consisting of two sets (W, U), viz., a set of vertices and a set of directed arcs which connect the vertices together; let us establish a correspondence between mathematical formulas and the graph.

A computational statement representing a formula specifies the generation of a set of data by means of a defined transformation upon the elements of another set of data. Let the input set be s_i and let s_0 be the set into which s_i is mapped through the trans-

* Department of Engineering. This work was supported by the Atomic Energy Commission, Division of Research (AT (11-1) Gen 10) and the Office of Naval Research, Information Systems Branch (Nonr 233(52)).

formation, f. Then we can write

$$s_0 = f(s_i), \tag{1}$$

and we usually state that s_0 is a function of s_i. The formula (1) is represented graphically in Figure 1. If w_i represents an initial input operation, then the source of s_i is unambiguous. If w_i represents an interior operation in the graph, the input set for w_f may have been generated by w_i or its predecessors and transported to accessible storage until required. Under any conditions, the presence of the arc from w_i to w_f unambiguously establishes that the operation (computations) represented by w_f cannot be executed until w_i has generated s_i.

Vertex Input Logic

Since the computational tasks (vertices) represent functions whose arguments are either conveyed along the arcs incident into a given vertex or were previously stored in an accessible location, it is possible to specify when a given vertex may "begin" by writing a Boolean expression whose truth value indicates that all precedence conditions have been satisfied. There are three types of "vertex-input" logic described in what follows.

Conjunctive Input. Let a given vertex require *all* of several sets of data, each set coming from a different source (vertex). Let the availability of these sets of data be represented by Boolean variables, $a_1, a_2, \cdots, a_n$. Then the event "all data available" is represented by the truth value of the conjunction $a_1 \wedge a_2 \wedge, \cdots, \wedge a_n$.

Mutually Exclusive Inputs. Let a given vertex require only one set of input data, but let this set of data be generated from one of several mutually exclusive origins (vertices). Let the event "all data available" be represented by the Boolean variable, a_1, and its alternate forms by $a_{11}, a_{12}, \cdots, a_{1m}$. Then the condition for vertex initiation is

$$a_1 = a_{11} \oplus a_{12} \oplus \cdots \oplus a_{1m}, \tag{2}$$

under the condition that all the a_{ij} are mutually exclusive events.

Compound Input. If the initiation of a vertex requires *several* sets of data, each of which has several mutually exclusive origins, we are interested in the truth value of the expression $(a_{11} \oplus \cdots \oplus a_{1m}) \wedge \cdots \wedge (a_{n1} \oplus \cdots \oplus a_{nm_n})$.

For purposes of calculation it is convenient to decompose vertices with compound input logic into several vertices with simple, i.e., only $\wedge$ or $\oplus$, input logic. This is easily done by introduction of an appropriate number of pseudo-vertices as shown in Figure 2.

Whenever a computation is described in terms of INCLUSIVE OR input conditions

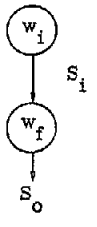

FIG. 1. The operation represented by w_i generates S_i; the operation (function) represented by w_f "transforms" S_i into S_0.

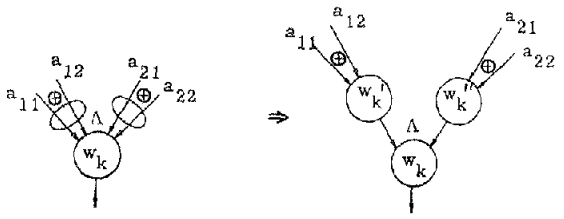

FIG. 2

we require a transformation such that only $\wedge$, $\oplus$ input conditions exist and thereby avoid complex definition of data distributions in the model.

Vertex Output Logic

The existence of mutually exclusive alternate origins of sets of data is due to the presence of *branching* or *decision* vertices in the graph. An arc incident out from a branching vertex may be traversed with a probability less than unity and may be selected conditional upon data being generated. In the usual language of computer programming such an operation is called a conditional transfer.

A branching vertex is defined as a vertex with EXCLUSIVE OR output logic as follows: If B is a Boolean variable representing the event that there is an output from a vertex and $b_1, b_2, \cdots, b_n$ are the Boolean variables representing *mutually exclusive events* at the output of the branching vertex, we can write

$$B \equiv b_1 \oplus b_2 \oplus \cdots \oplus b_n, \tag{3}$$

where the two sides of the expression are logically equivalent [6].

Program flow may also occur from a given vertex along several different arcs simultaneously, i.e., there may be a bundle of arcs, each with the same origin vertex and different terminal vertices, that are incident out from a vertex (or pseudo-vertex) jointly. Then, if for any event represented by b_i as above, we define Boolean variables $b_{i1}, b_{i2}, \cdots, b_{im}$ such that there is a logical equivalence between any pair of b_{ij}, we can unambiguously denote the simultaneity of output arcs on the graph. We choose to use the symbol, $*$, to mark the logical equivalence of events on output arcs and obtain

$$b_i \equiv b_{i1} * b_{i2} * \cdots * b_{im} \tag{4}$$

as a representation of simultaneous vertex output events. The $*$ output condition is sometimes referred to as an AND output condition in this paper.

If the sets $\{b_{j1}, b_{j2}, \cdots, b_{jm_j}\}$, $j = 1, 2, \cdots, n$, represent all of the possible output events that can occur upon completion of a vertex, then the occurrence of a compound output event will be represented by the expression

$$(b_{11} * \cdots * b_{1m_1}) \oplus \cdots \oplus (b_{n_1} * \cdots * b_{nm_n}). \tag{5}$$

As in the case of compound input logic, pseudo-vertices are introduced to decompose compound output logic into simple output logic. This is illustrated in Figure 3 for the condition where w_k initiates w_a, w_b, w_c, w_d under the conditions $(w_a * w_b) \oplus (w_c * w_d)$.

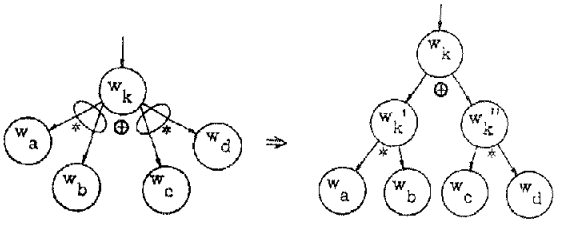

FIG. 3

Let us consider the probabilities associated with branching arcs. If w_i and w_j are origin and terminal vertices of an arc, u_{ij}, let the probability of reaching w_j via u_{ij} be q_{ij}. In general, q_{ij} is a conditional probability of traversing u_{ij} given that w_i has been reached. We now require that program flow be "conserved," i.e., if $u_{k1}, u_{k2}, \cdots, u_{kn}$ are the mutually exclusive arcs incident out from a branching vertex w_k with simple EXCLUSIVE OR output logic, then

$$\sum_{i=1}^{n} q_{ki} = 1. \tag{6}$$

It is instructive to compare our graphical network with two types of networks used in other contexts. The first is the PERT [7] network, in which all vertices have AND input and output logic. Hence all vertices are reached with probability one. Another special network is the graph representation of the sequential machine, where vertices correspond to the internal states of the machine and arcs represent the transitions between states. All vertices in the latter cases have EXCLUSIVE OR input and output logic and the arc traversal probabilities are the state-to-state transition probabilities. The probabilistic properties of such a network can be represented as a simple Markov chain. Our network is a more general form of these two cases and the computational probabilities are more difficult to determine.

Vertices

A computational vertex on a directed graph represents an unambiguously defined computational statement. The execution of the statement may imply the execution of several other more elementary operations or "microstatements." A certain amount of "fine structure" may not explicitly appear in the graph dependent upon the relative complexity of functions represented by single vertices. In fact the fine structure implied by any particular vertex might itself be represented by a graph whose characteristics are identical to those discussed above. Hence if the assumption is now made that vertex properties (such as time required for the operation) are known, then treatment of the problem of representing collections of vertices by a single vertex [2] will indicate the effect of the implied fine structure.

Vertex Probabilities

In the following two procedures are considered for determining the probability, p_k, of ever reaching a given vertex, w_k, on the graph. It is assumed that the directed graphs are acyclic, that all vertices possess simple input and output logic and that

all branching decisions are independent. A properly connected graph is defined as one in which $0 < p_k \leq 1$ and p_k is just the probability of w_k being linked to a subset of the set of origin vertices which have no logical predecessors.

Vertex Probability Computational Procedure I

A partially ordered set of computations represented on a directed graph can be regarded as a collection of mutually exclusive subgraphs whose vertices possess no exclusive-OR input or output logic within any subgraph. These subgraphs we call AND-type subgraphs. Each branching vertex can be regarded as a multiway switch in which one and only one position (emergent arc) is selected each time the vertex is executed. Each switch position is chosen according to the set of arc traversal probabilities assigned to the arcs incident out from a given branching vertex.

If any given graph could be partitioned into a number of mutually exclusive subsets (AND-type subgraphs) whose union was equivalent to the set represented by the graph, then the following procedure would determine the probability of ever reaching a vertex w_i.

1. For each AND-type subgraph establish the arc traversal probabilities of all arcs contained in it.
2. Compute the probability of traversing each AND-type subgraph as the product of the arc traversal probabilities found in step 1.
3. Consider each vertex in turn and find the set of AND-type subgraphs containing that vertex. The probability of ever reaching that vertex is the sum of the probabilities computed for the subgraphs in step 2.

We are left with the need for a systematic procedure to find all the distinct AND-type subgraphs into which a computational network can be partitioned. It is helpful to introduce two structural indices associated with the vertices, called the precedence and ante numbers.

A connection matrix $[Z]$ describes graph linkages with a nonzero Z_{ji} entry whenever there is an arc u_{ij} from vertex w_i to vertex w_j. From $[Z]$ is obtained [2, 3, 5] a square Boolean precedence matrix, $[D]$, which has dimensions equal to the number of vertices, contains nonzero entries in its kth row to mark the predecessor (not necessarily immediate) vertices of vertex w_k and contains nonzero entries in its kth column to mark the successor (not necessarily immediate) vertices of vertex w_k. We define

$$d_k \triangleq \sum_{i=1}^{N_w} d_{ki} \tag{7}$$

as the precedence number of w_k and

$$f_k \triangleq \sum_{i=1}^{N_w} d_{ik} \tag{8}$$

as the ante number of w_k, where d_{ij} is an element of $[D]$ and N_w is the number of vertices in the graph. The precedence and ante numbers of a given vertex have certain properties that are simply related to the partial orderings between vertices, viz., if w_i precedes w_j then $d_i < d_j$ and $f_i > f_j$, but not conversely.

Now let S be the set of branching vertices in the graph and execute the following procedure.

1. Find the set of branching vertices S.
2. Let the set of (branching) arcs incident out from each $w_i \in S$ be U_i^+, and let I_i be the index set of the arcs in U_i^+, i.e., $u_j \in U_i^+ \Leftrightarrow j \subset I_i$. Note that u_j just represents an indexing of the branching arcs emerging from a vertex w_i. We imply that we retain a mapping between the u_j and the previously described u_{ij}. Now form a distinct combination of indices $C(k)$ (the kth such distinct combination, say) by selecting one index from each of the index sets I_i, $w_i \in S$, and form the union set of arcs

$$V_k \triangleq \bigcup_{j \in C(k)} u_j , \tag{9}$$

where V_k will be used to form an AND-type subgraph as follows.
3. Find the set of arcs

$$E_k \triangleq \bigcup_{w_i \in S} U_i^+ - V_k , \tag{10}$$

i.e., the set of branching arcs which are *excluded* from the AND-type subgraph being formed.
4. We must now "purge" the graph, i.e., remove those vertices and arcs whose presence in the graph is precluded by the removal of arcs in E_k from the graph. To accomplish this with an iterative procedure we define $R_k(-)$ as the set of arcs and vertices removed from the graph during formation of the subgraph. Initially, $R_k(-) = E_k$. Define W_n as the set of vertices with precedence number n. Now, in order of increasing precedence number, examine all the vertices in the (original) graph as indicated in the steps that follow.
5. Examine the vertices in the sets W_0, W_1, W_2, etc., *in that order*, and determine their input logic. Let w_i be the vertex under examination. Determine the input logic of w_i :
   a. w_i *has* AND *input logic.* Find the set of arcs incident into w_i, viz., U_i^-. If any of the arcs in U_i^- are also in $R_k(-)$, add $w_i \cup U_i^- \cup U_i^+$ to $R_k(-)$, i.e., w_i and the arcs both incident into and out from w_i. If $U_i^- = \phi$, i.e., $w_i \in W_0$, no additions are made to $R_k(-)$.
   b. w_i *has* EXCLUSIVE OR *input logic.* If *all* the arcs in U_i^- are in $R_k(-)$, then add $w_i \cup U_i^+$ to $R_k(-)$. If not all the arcs in U_i^- are in $R_k(-)$, i.e., $U_i^- \not\subset R_k(-)$, then there should be one and only one arc in U_i^- not in $R_k(-)$. If there are more, then the input logic at w_i is not EXCLUSIVE OR, and the graph has improper logical structure. If there is properly only one arc U_i^- not in $R_k(-)$, $R_k(-)$ is unchanged.
6. When the complete set W has been exhausted, the vertices and arcs that have not been removed from the original graph, i.e., the set $(W, U) - R_k(-)$, constitute the AND-type subgraph that results from a choice of branching arcs whose indices are in the set $C(k)$.

All the other AND-type subgraphs into which (W, U) can be partitioned are found by repeating the above procedure using all other distinct combinations $C(k)$ defined in step 2 above. It is worth noting here that the procedure given above for finding the AND-type subgraphs into which a given acyclic directed graph can be partitioned depends not only upon the structure of the original graph but more importantly upon the input and output logic possessed by the vertices. Hence even though we select structurally distinct sets V_i and V_j, $V_i \neq V_j$, the AND-type subgraph completion procedure above may yield the same AND-type subgraph by the discarding of subgraphs reached with probability zero but containing index selected branching arcs. This point is illustrated by Figure 4, in which selection of $V \triangleq \{u_{12}, u_{34}\}$ and $V' \triangleq \{u_{12}, u_{35}\}$ yields only one AND-type subgraph $G = G' = \{w_1, w_2, w_6, u_{12}, u_{26}\}$. Furthermore, it must be noted that the above procedure is essentially enumerative and therefore the size of the problem must be considered. An upper bound for the number N of AND-type subgraphs obtainable from a graph containing

n branching vertices with m_i arcs incident out from the ith branching vertex $i = 1, 2, \cdots, n$ is

$$\bar{N} = \prod_{i=1}^{n} m_i . \tag{11}$$

The small example illustrated in Figure 5 indicates that $\bar{N}$ is generally much less than the upper bound. However, the complexity with which it might be possible to deal using the above enumerative procedure seems so small that we now leave aside this procedure and consider a nonenumerative procedure based on a more restrictive assumption about the original graph.

Vertex Probability Computational Procedure II

A nonenumerative procedure becomes possible if a certain amount of semantics is introduced into the process of generating a precedence matrix which picks out select predecessors and successors only if they also satisfy logical conditions and maintain proper connectivity. The following discussion attempts to clarify these remarks.

Consider a vertex w_k in the interior of a graph. w_k is reached from a subset of the origin vertices via a subgraph consisting of w_k, its logical predecessors and associated connecting arcs. We choose to distinguish at this point between *structural* predecessors denoted D_k^- (defined by nonzero entries in the kth *row* of the matrix $[D]$) and the set of *logical* predecessors, denoted E_k^-, which are a subset of D_k^- that are reached with a nonzero probability given that w_k has been reached. In similar vein, topological and logical *successor* sets D_k^+ and E_k^+ are defined. To clarify further it must be recognized that the subgraph consisting of the set of vertices $D_k^- \cup w_k$ and the associated connecting arcs can be partitioned into a number of distinct AND-type subgraphs. The condition for elements of the E_k^- sets is then equivalent to stating that a logical predecessor of w_k must be included in at least one of the distinct AND-type subgraphs into which $D_k^- \cup w_k$ and the associated connecting arcs is partitioned. A similar condition holds between E_k^+ and $D_k^+ \cup w_k$. It follows that

$$E_k^- \subseteq D_k^-, \qquad E_k^+ \subseteq D_k^+. \tag{12}$$

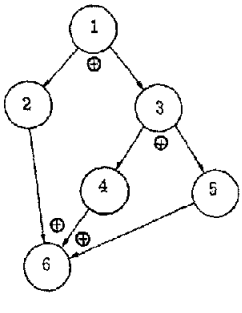

FIG. 4. A graph illustrating the nonuniqueness of AND-type subgraph selection

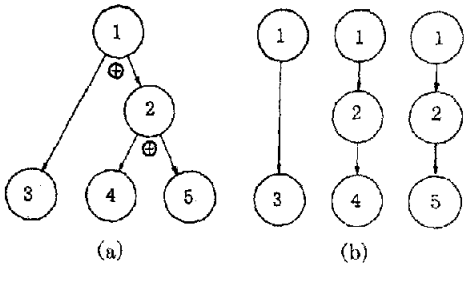

FIG. 5. An example graph: a, original graph; b, partition

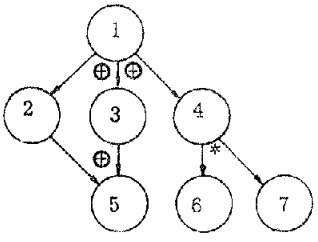

FIG. 6. A graph in which $D_k^{\pm} = E_k^{\pm}$

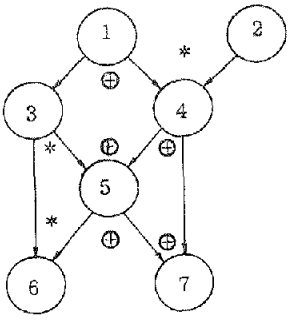

FIG. 7. A graph in which $D_k^{\pm} \neq E_k^{\pm}$

For illustration, consider the directed graph in Figure 6 where we have

$$D_5^{-} = \{1, 2, 3\}, \qquad E_5^{-} = \{1, 2, 3\}, \tag{13}$$

and Figure 7 where we have

$$D_6^{-} = \{1, 2, 3, 4, 5\}, \qquad E_6^{-} = \{1, 3, 5\}. \tag{14}$$

One very important difference between $D_k^{\pm}$ and $E_k^{\pm}$ lies in their determination. The sets $D_k^{\pm}$ can be obtained directly from the precedence matrix $[D]$. $E_k^{\pm}$ may require the enumeration of the AND-type subgraphs into which $D_k^{\pm} \cup w_k$ and the associated connecting arcs are partitioned.

In the remainder of this study, we concern ourselves with directed graphs for which $E_k^{\pm} = D_k^{\pm}$, $w_k \subset W$, a condition which holds for all the complex graphs encountered in our experiments [3] and which may hold always if the logic implied by the original computational formulas is retained in defining a graph and no artificial graph linkages are arbitrarily inserted.

With the above discussion in mind we proceed to describe a practicable algorithm for computing vertex probability.

Consider a subgraph G_k^{-} incident into a vertex w_k and including w_k. The probability, p_k, of ever reaching w_k depends upon the traversal probabilities of arcs incident out from branching vertices in G_k^{-}. We find the latter formulation useful whenever w_k has conjunctive input logic. In the case that w_k has disjunctive input logic we note that the probability of the union of a number of mutually exclusive events is equal to the sum of their individual probabilities of occurrence.

If we make use of the above observations and the assumptions that branching decisions are mutually independent and $D_k^{\pm} = E_k^{\pm}$, the algorithm for computing the probability of ever reaching a vertex w_k follows.

1. Examine in order of subscript the vertex sets w_i, $i = 0, 1, 2, \cdots$, where i is the precedence number. Let w_k be a vertex under consideration.
2. Find Z_k^{-}, the set of immediate predecessors of w_k. Four cases can occur:
   a. $Z_k^{-} = \phi$, i.e., w_k is an initial vertex and $p_k = 1$.
   b. Z_k^{-} consists of a single vertex, e.g., $Z_k^{-} = \{w_a\}$. Then $p_k = p_a q_{ak}$.
   c. Z_k^{-} consists of more than one vertex and w_k has exclusive-OR input logic. Then

$$p_k = \sum_{w_i \in Z_k^{-}} p_i q_{ik}. \tag{15}$$

d. Z_k^- consists of more than one vertex and w_k has AND input logic. We deal with a subset of D_k^-, S_k^-, which contains only the branching vertices preceding w_k. Then

$$p_k = 1 - \sum_{w_i \in S_k^-} p_i \sum_{w_j \in Z_i^+ - \cap D_k^-} q_{ij} . \tag{16}$$

Computational Experiments

Figures 8–12 depict graphs representing computations arising in X-ray crystallography (Figure 8), Numerical Weather Prediction (Figures 9, 10) and the Assignment and Sequencing Computation (Figures 11, 12). Cyclic to acyclic transforma-

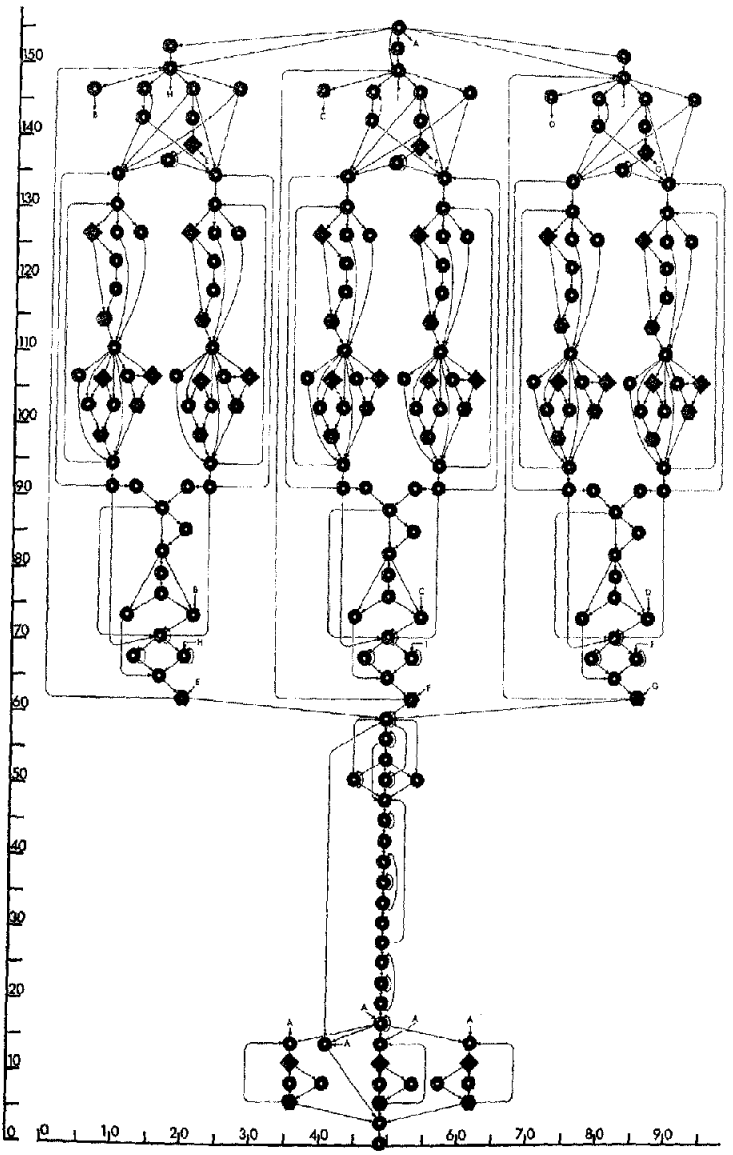

FIG. 8. X-Ray

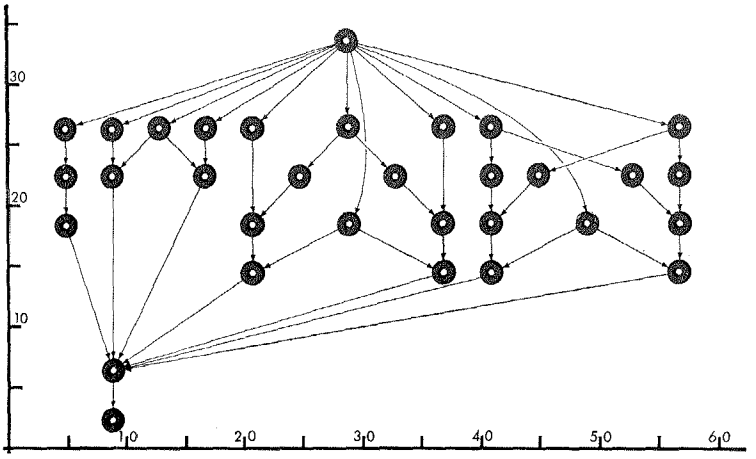

FIG. 9. NWP32

tions [3] remove all feedback arcs leaving the nonfeedback topology unchanged but primarily affecting the estimated operation times. Instead of explicitly labeling the input and output control conditions, the vertex shapes are varied as follows:

1. Circle: AND INPUT, AND OUTPUT
2. Diamond: AND INPUT, OR OUTPUT
3. Hexagon: OR INPUT, AND OUTPUT

The OR input-OR output condition did not explicitly appear in the graphs studied. The coordinates are merely a convenience for locating vertices. Table 1 is a summary of graph statistics for the computations modeled as vehicles for assignment and sequencing experiments which used the vertex probability calculations.

The results of these computations then serve as inputs to programs which are used in assignment and sequencing of operations on computers and estimating the resulting expected computation time [3]. Figures 13–15 illustrate the preparation of graph description (LINK 1) and the assignment and sequencing perturbation process (LINK 2). The vertex probability computation occurs in the former.

Dependent Branching Decisions

Thus far, the vertex computational probability algorithm has been derived on the assumption that all the branching decisions executed in the computational network were mutually independent. We now modify our algorithm to include the case where the branching decisions are not mutually independent.

First of all, it would be instructive to give a couple of instances where nonindependence of branching decisions arises. Consider the following ALGOL statements:

L1: **if** $a_1 \leq x \wedge x \leq a_2$ **then go to** L2 **else go to** L3;
L2: **if** $a_3 \leq y \wedge y \leq a_4$ **then go to** L4;
L3: ⋯
⋮
L4: ⋯

These statements may be represented on a directed graph, with decision vertices represented by diamond-shaped boxes, as shown in Figure 16. Now if x and y are

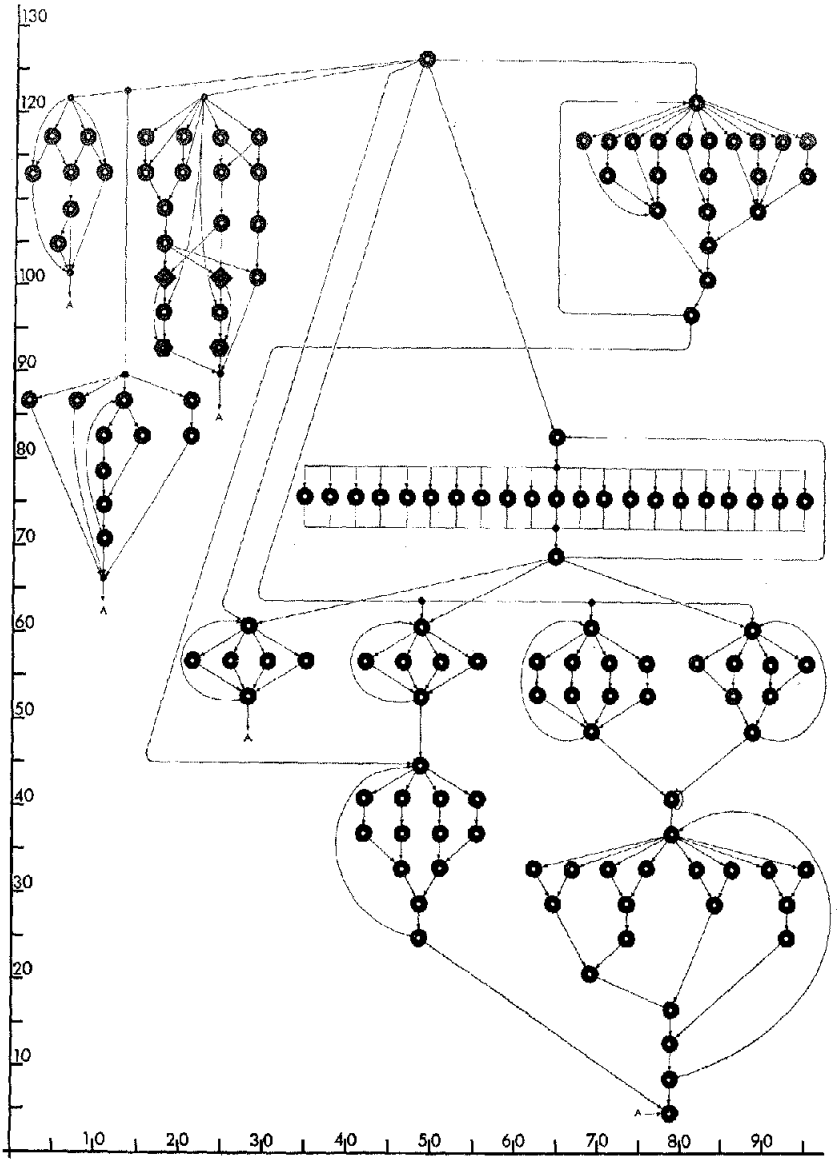

FIG. 10. NWP147

independent of each other, i.e., their values are not linked computationally or otherwise, then the arc traversal probability $q_{L2,L4}$ is determined independently of $q_{L1,L2}$. On the other hand, if x and y are computationally related, there is a conditional probabilistic relation between $q_{L2,L4}$ and $q_{L1,L2}$, determined by the relation between x and y, i.e., by the probability distributions of x and y over the intervals $[a_1, a_2]$ and $[a_3, a_4]$, respectively. In particular, let $y = Tx$, where T is a computation that maps x into y, and let $f(x)$ be the probability density function of x on the interval $[a_1, a_2]$. Then, knowing the relation $y = Tx$, we can determine the region, i.e., the union set of disjoint intervals, into which the interval $[a_1, a_2]$ maps. Although it is an abuse of notation, let this union set be denoted by $[a_1', a_2']$. We can

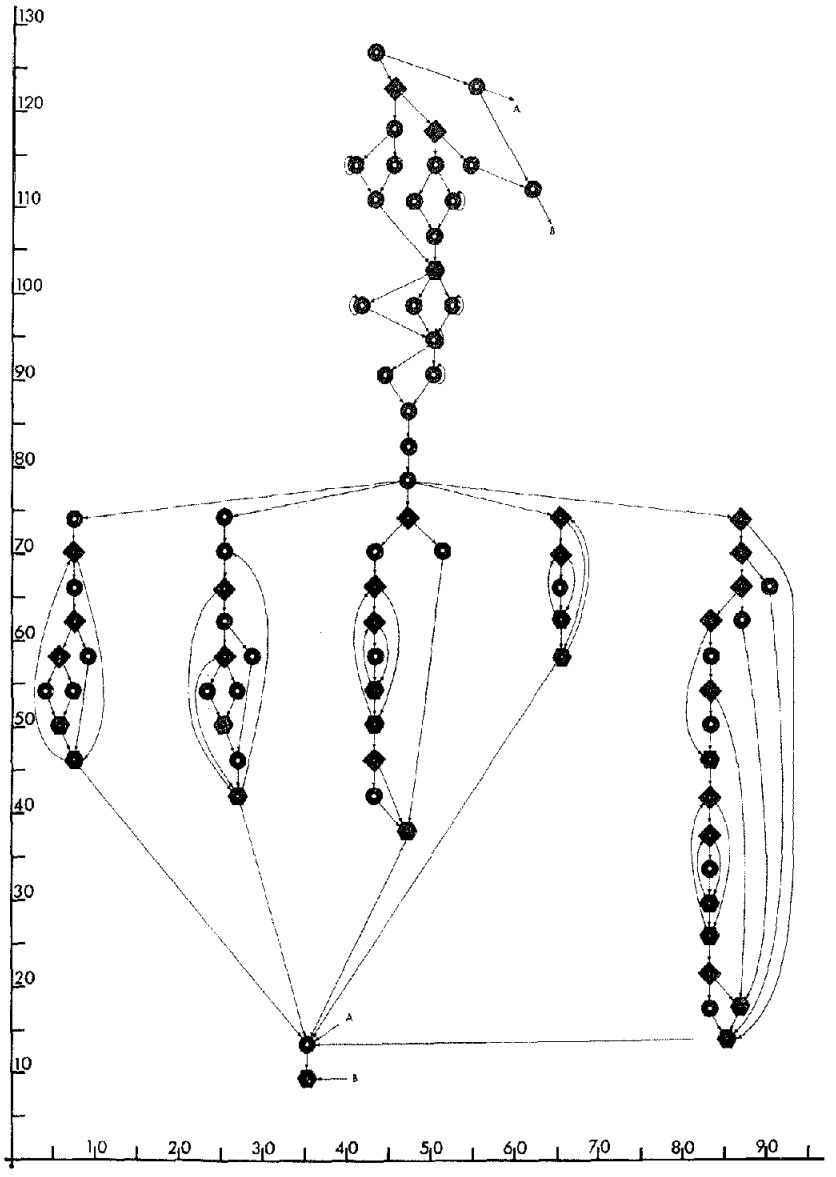

FIG. 11. 82V

then determine the probability density function of $y = Tx$ over $[a_1', a_2']$, and hence also the density function of y on the intersection set $[a_1', a_2'][a_3, a_4]$. From this information, we can finally determine the conditional probability for $(y \in [a_3, a_4] \mid x \in [a_1, a_2])$, i.e., the probability that $y \in [a_3, a_4]$ given $x \in [a_1, a_2]$. Now if the branching decisions made in vertices $L1$ and $L2$ were statistically independent, the probability of executing vertex $L4$ would be

$$p_{L4} = p_{L1}q_{L1,L2}q_{L2,L4} \qquad \text{(independence)}. \quad (17)$$

However, if these branching decisions were *not* statistically independent (as we

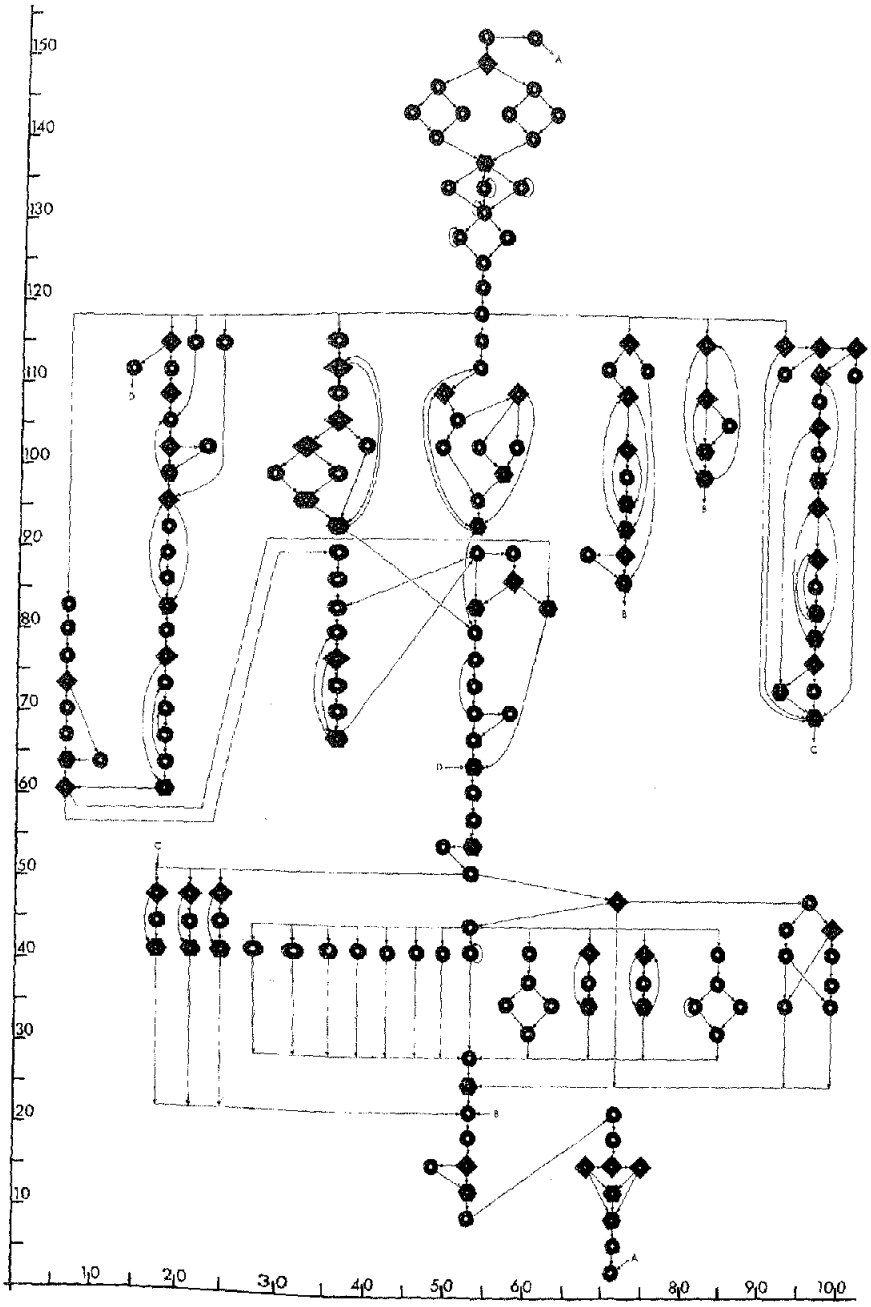

FIG. 12. L2

have postulated), then

$$p_{L4} = p_{L2}q_{L1,L2}(q_{L2,L4} \mid q_{L1,L2}) \qquad \text{(dependence)}, \quad (18)$$

where $(q_{L2,L4} \mid q_{L1,L2})$ is the conditional probability of traversing arc $(L2, L4)$ given that arc $(L1, L2)$ has been traversed. The conditional probability $(q_{L2,L4} \mid q_{L1,L2})$ can be determined from the conditional probability for $(y \in [a_3, a_4] \mid x \in [a_1, a_2])$.

TABLE 1. GRAPH STATISTICS

GRAPH STATISTICS Statistic	GRAPH									
	NWP32		82V		NWP147		L2		XRAY	
No. of Vertices	32		82		147		193		223	
No. of Vertex Clusters	32		74		147		176		202	
No. of Arcs	47		129		246		294		413	
No. of Feedback Arcs (cycles)	0		14		12		24		45	
No. of Vertices with										
OR input logic	0		16		2		33		24	
OR output logic	0		21		2		40		24	
AND input logic	32		66		145		160		199	
AND output logic	32		61		145		153		199	
*	WFB	NFB	WFB	NFB	WFB	NFB	WFB	NFB	WFB	NFB
Avg. no. arcs input to any vertex	1.47	1.47	1.57	1.40	1.67	1.59	1.52	1.40	1.85	1.65
Max. no. of arcs input to any vertex	7	7	6	6	21	21	12	12	5	4
Avg. no. of arcs output from:										
any vertex	1.47	1.47	1.57	1.40	1.67	1.59	1.52	1.40	1.85	1.65
vertex w/AND output logic	---	1.47	---	1.18	---	1.59	---	1.22	---	1.61
vertex w/OR output logic	---	---	---	2.05	---	2.00	---	2.07	---	2.00
Max. no. of arcs output from any vertex	11	11	5	5	25	25	12	12	11	11

* WFB: includes feedback arcs; NFB: does not include feedback arcs.

Let us give one more instance where the simplifying assumption of statistical independence is incorrect. Consider the graph in Figure 17. The branching decisions in vertices 1 and 2 are made in parallel. Let $p_1 = p_2 = 1$, and let us compute p_3. Now the branching decisions classify the same datum, x, into the two classes c_1 and c_2, and we wish to determine the probability that $x \in c_1$ *and* $x \in c_2$. This probability clearly depends upon whether c_1 and c_2 do not intersect ($c_1 \cap c_2 = \phi$), partially interesect ($c_1 \cap c_2 \neq \phi$) or totally intersect (either $c_1 \cap c_2 = c_1$ or $c_1 \cap c_2 = c_2$). If $c_1 \cap c_2 = \phi$, it is clear that $p_3 = 0$ (this case would not be properly representable by the graph in Figure 17, since w_3 would be redundant). On the other hand, if $c_1 \cap c_2 \neq \phi$, then $(q_{23} \mid q_{13}) \neq q_{23}$. The conditional probability $(q_{23} \mid q_{13})$ is determined from a knowledge of the probability density functions of x on c_1 and c_2, and the relation between c_1 and c_2. Specifically,

$$(q_{23} \mid q_{13}) = \begin{cases} 0, & c_1 \cap c_2 = \phi, \\ (q_{23} \mid q_{13}), & c_1 \cap c_2 \neq \phi, \\ 1, & c_1 \cap c_2 = c_1(c_1c_2), \\ q_{23}/q_{13}, & c_1 \cap c_2 = c_2(c_2c_1), \end{cases} \tag{19}$$

and hence, since $p_3 = q_{13}(q_{23} \mid q_{13})$,

$$p_3 = \begin{cases} 0, & c_1 \cap c_2 = \phi, \\ q_{13}(q_{23} \mid q_{13}), & c_1 \cap c_2 \neq \phi, \\ q_{13}, & c_1 \cap c_2 = c_1, \\ q_{23}, & c_1 \cap c_2 = c_2. \end{cases} \tag{20}$$

In cases where there are more than two branching decisions that are dependent upon one another, expressions similar to the ones above can be written, except

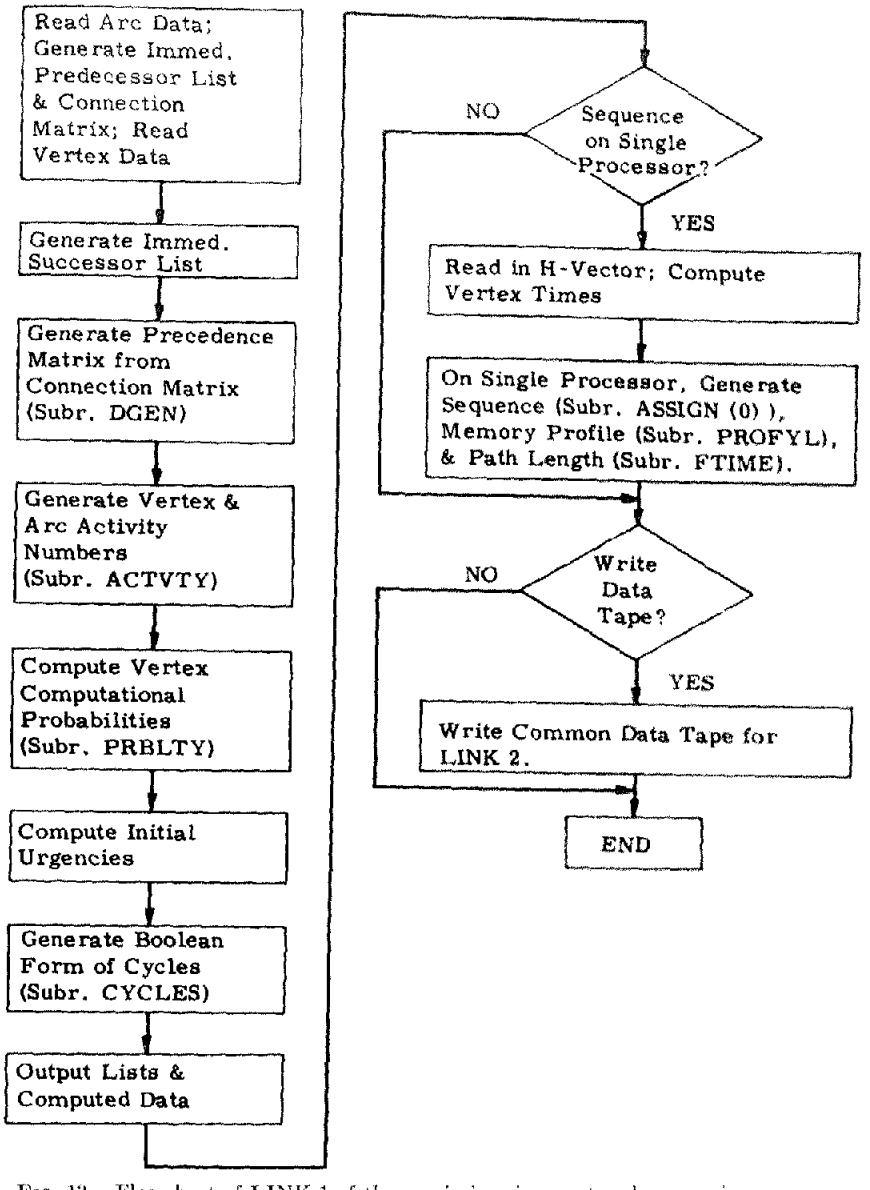

FIG. 13. Flowchart of LINK 1 of the a priori assignment and sequencing program

that the number of different cases might become large. Generally speaking, if we are concerned with n dependent branching decisions, then expressions of the form

$$q_1(q_2 \mid q_1)(q_3 \mid q_2 q_1) \cdots (q_n \mid q_{n-1} \cdots q_1) \tag{21}$$

will arise where the q_i are arc traversal probabilities.

Let us now address ourselves to the problem of incorporating these cases into the vertex computational probability algorithm. Let us begin by assuming that we have chosen a particular vertex w_k, found the sets D_k^- and S_k^- and chosen the

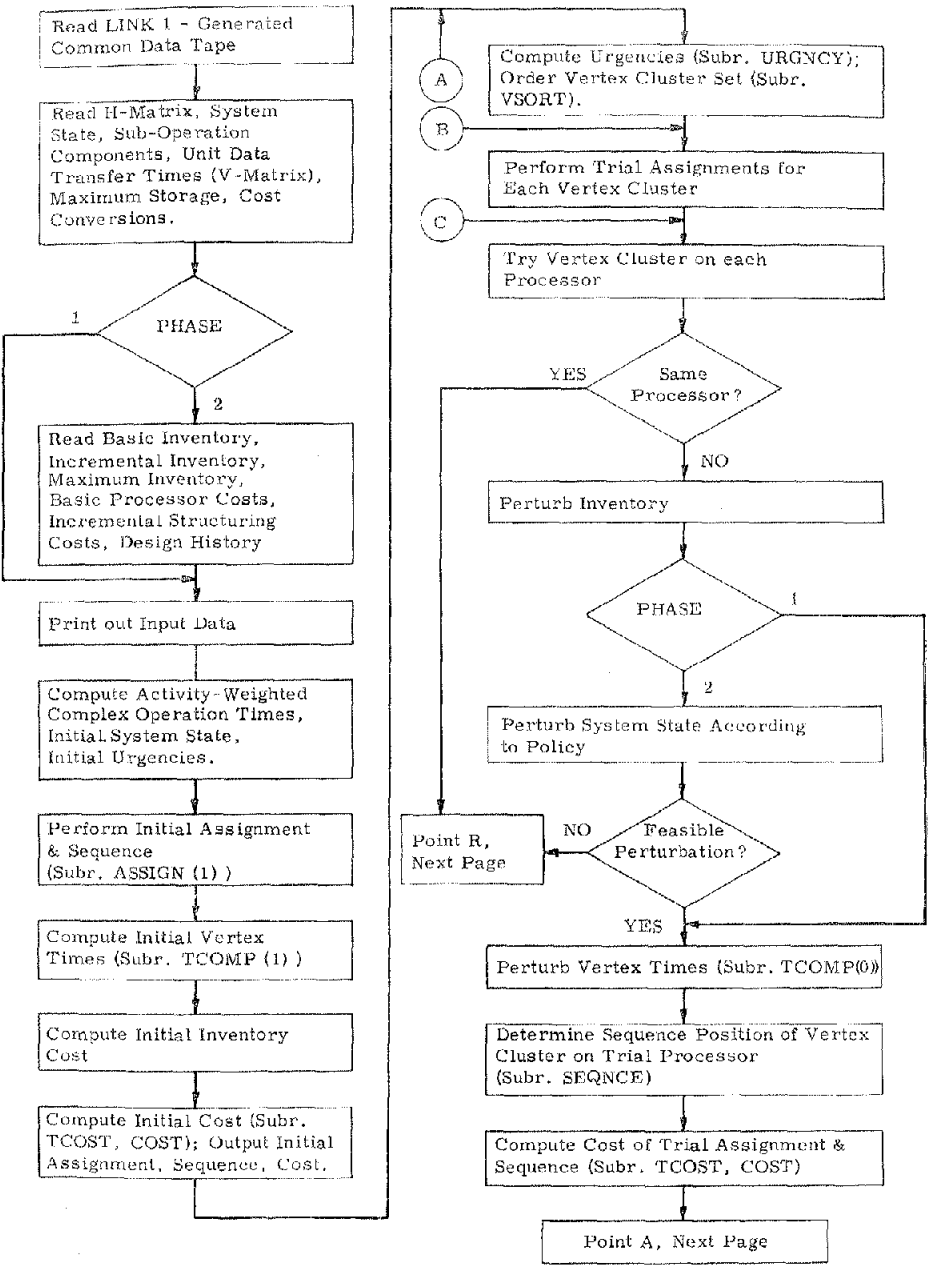

FIG. 14. Flowchart of LINK 2 of the a priori assignment and sequencing program

ith mutually exclusive AND-type subgraph. The probability that w_k will be connected to the origin vertices by means of the ith AND-type subgraph can be regarded as the probability of the joint occurrence of a number of mutually dependent binary-valued events, i.e., *dependent* Bernoulli trials. If $\{u_1, u_2, \cdots, u_{m_i}\}$ is the set of arcs in the ith AND-type subgraph and $\{q_1, q_2, \cdots, q_{m_i}\}$ is the set of corresponding arc

traversal probabilities, then the probability that all the arcs will be traversed is

$$p_{ki} = q_1(q_2 \mid q_1) \cdots (q_{m_i} \mid q_{m_{i-1}} \cdots q_1). \tag{22}$$

Now those conditional arc traversal probabilities that correspond to arcs incident

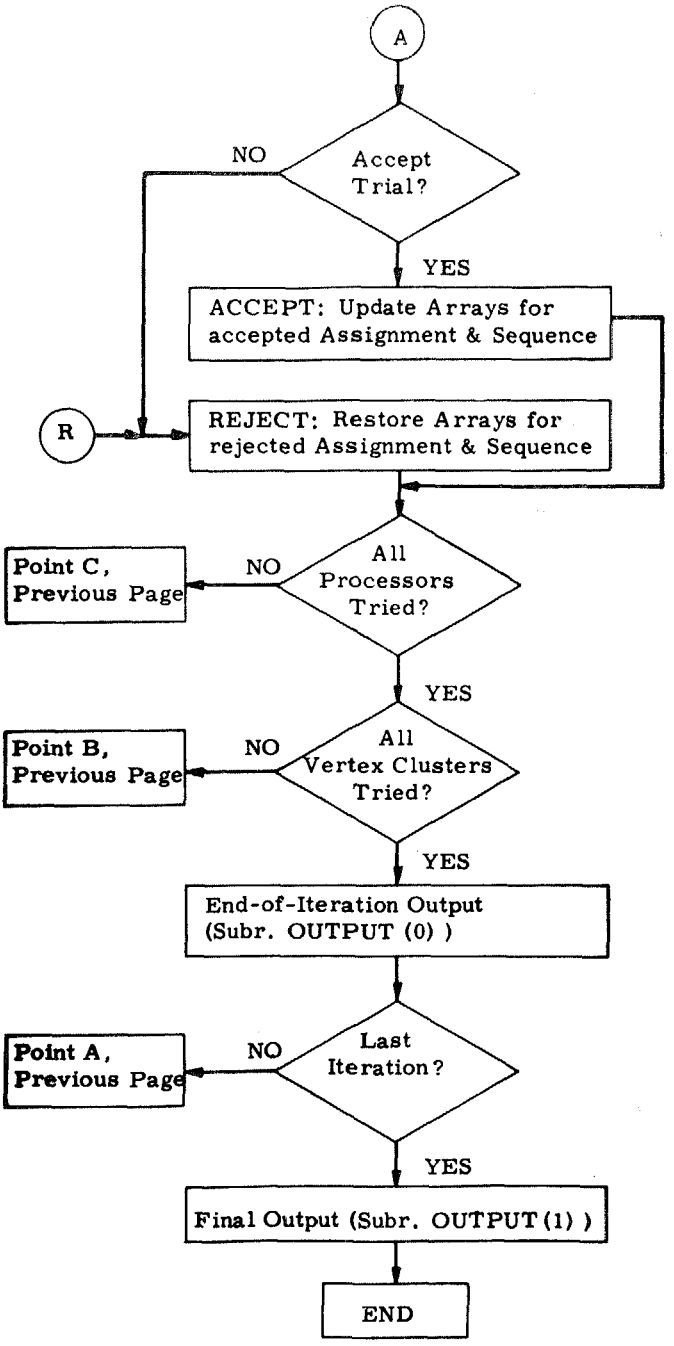

FIG. 15

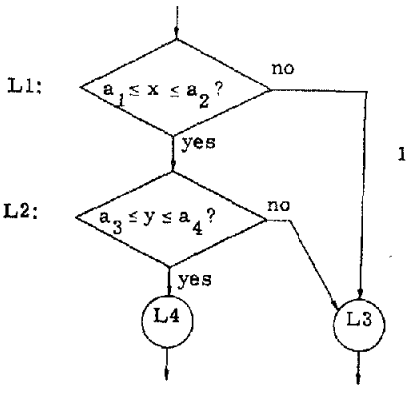

FIG. 16

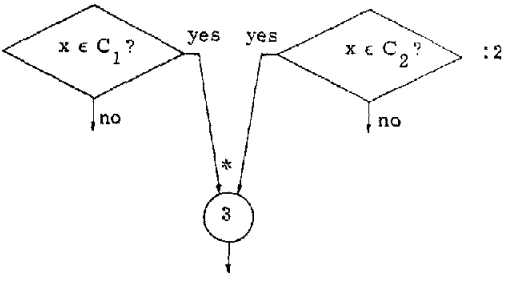

FIG. 17

out from nonbranching vertices represent events that occur with certainty, and hence they are really not conditional upon the outcome of any of the branching decisions. Thus, as before, these unity probabilities need not be included in the formula for p_{ki}. Hence,

$$p_{ki} = q_1(q_2 \mid q_1) \cdots (q_{m_{i'}} \mid q_{m_{i-1}'} \cdots q_1), \tag{23}$$

where $\{u_1, u_2, \cdots, u_{m_{i'}}\}$ represents the set of arcs incident out from branching vertices in the ith AND-type subgraph. Having computed all the p_{ki}, we may obtain p_k through

$$p_k = \sum_i p_{ki}. \tag{24}$$

The foregoing algorithm is an extension of the enumerative vertex computational probability algorithm previously derived, and it presumes that sufficient information is available for the evaluation of all the required conditional arc traversal probabilities.

Unfortunately, due to the conditional relations between the various arc traversal probabilities, our extended algorithm cannot be recast into a more compact form as was done when the branching decisions were mutually independent. Hence, the extended algorithm remains essentially enumerative.

Conclusion

This paper has formulated procedures for determining the probability of reaching vertices in a transitive directed graph representation of computations and has discussed a number of problems arising in such modeling.

The algorithms are essential to methods for a priori estimation of computation time on models of computer systems and have proven themselves effective in a number of experimental studies.

Further work is needed to handle branching dependency, to automatically generate estimates of arc traversal probabilities from initial formulas or programs and to test for improperly connected graphs.

Other papers will deal with cyclic to acyclic transformations, path length calculations and experiments in automatic assignment and sequencing.

REFERENCES

1. BERGE, C. *The Theory of Graphs and Its Applications*. John Wiley and Sons, New York, 1962.
2. ESTRIN, G., AND TURN, R. Automatic assignment of computations in a variable structure computer system. *IEEE Trans. EC-12* (Dec. 1963), 755–773.
3. MARTIN, D. The automatic assignment and sequencing of computations on parallel processor systems. Ph.D. thesis, U. of California, Los Angeles, Jan. 1966.
4. ELMAGHRABY, S. E. An algebra for the analysis of generalized activity networks. *Manage. Sci. 10* (April 1964), 494–514.
5. RUSSELL, E. C. Automatic assignment of computational tasks in a variable structure computer. M.S. thesis in Engineering, U. of California, Los Angeles, 1963.
6. CARNAP, R. *Introduction to Symbolic Logic and Its Applications*. Dover Publications, New York, 1958, pp. 19–23.
7. KELLEY, J. E., JR. Parametric programming and the primal-dual algorithm. *Oper. Res. 7* (May–June 1959), 327–334.

RECEIVED MARCH, 1966; REVISED MAY, 1966