# DEFINITIONS AND PROCEDURES EMPLOYED

in the

# GERT EXCLUSIVE-OR PROGRAM

Research Sponsored

by

Electronics Research Center
National Aeronautics and Space Administration
NASA Research Grant NGR 03-001-034



Phillip C. Ishmael and A. Alan B. Pritsker

## ABSTRACT

This report describes the definitions of the variables and the techniques used in a digital computer program for analyzing GERT networks which contain nodes of the EXCLUSIVE-OR type and branches which have both a probability and a time associated with them. The time associated with a branch can be a random variable. The program calculates the probability, the expected time and the variance in the time to go from each source node of the GERT network to each sink node.

Programs have been written in FORTRAN II and FORTRAN IV and have been exercised on the IBM 1130, GE 225 and CDC 3400 computers. This report details the methods used in these programs. Emphasis has been placed on storage conservation. Methods for determining the loops and paths of a network are described. The equations necessary to calculate the values associated with the topology equation from loop and path values are presented. Flow charts and FORTRAN listings are given for each subprogram. An analysis of the size of each array is made and the relationships between dimensioned variables is discussed. A method for obtaining results for large networks by segmenting the network is presented at the end of this report.

The program described in this report has been submitted to COSMIC.

# DEFINITIONS AND PROCEDURES EMPLOYED
# in the
# GERT EXCLUSIVE-OR PROGRAM

## Introduction

### Purpose

A digital computer program which analyzes GERT networks containing only EXCLUSIVE-OR nodes was developed at the RAND Corporation. (1, pp. 81-95). A user's manual (2) for a modified version of the GERT EXCLUSIVE-OR node program details the operational procedures involved in using the program. This report presents the definitions and computer procedures used to analyze networks with EXCLUSIVE-OR nodes.

The modified program exists in three functionally identical versions: a FORTRAN II version which has been run on the GE 225 computer, a FORTRAN IV version (without logical variables) which has been run on the IBM 1130 computer, and a FORTRAN IV version which has been run on the CDC 3400 computer. The basic version that will be discussed here is the IBM 1130 version since it differs from the GE 225 version only in the input-output statements and from the CDC 3400 version only in the logical transfer statements.

### Background

The GERT EXCLUSIVE-OR program determines the source nodes, the sink nodes, the paths connecting the source nodes to the sink nodes, and the loops of a network. The standard output from the program includes appropriate problem identification headings, the paths and loops of the network, the probability of realizing a sink node from any source node, and the mean and variance of the time to realize a sink node, given that the sink node is realized and given an initial source node. The options exist to: 1) delete the loop and/or path output for large or complex networks; 2) delete loops with low probabilities of being realized; and 3) normalize the output if loops are deleted.

Input to the program includes appropriate problem identification information and the branches of the network. Information concerning each branch includes the start node and end node for the branch, the probability of realizing the branch, and a label to identify a moment generating function (M.G.F.). The M.G.F. is described by a three-letter code and the parameters of the M.G.F. The program determines all paths and loops of the network and their associated values based on the input information.

The values associated with the loops and paths of the network to obtain the desired output statistics are:

1. the probability;
2. the mean time; and
3. the second moment of the time.

The probability associated with the loop or path is the product of the probabilities of the branches comprising the loop or path. The expected time to traverse a loop or path is the sum of the expected times of the branches of the loop or path. The second moment of the time to traverse a loop or path is given by the following equation: (1, p. 83)

$$\mu_{2L} = \mu_{1L}^2 - \sum_{i \varepsilon L} \mu_{1i}^2 + \sum_{i \varepsilon L} \mu_{2i}$$

where

$\mu_{2L}$ = second moment of L where L represents the loop or path;

$\mu_{1L}$ = expected time to traverse L (first moment of time to traverse L);

$\mu_{1i}$ = first moment of time for loop or path i comprising L;

$\mu_{2i}$ = second moment of time for loop or path i comprising L; and

$i \epsilon L$ indicates that the summations are over all branches comprising L.

The above three values associated with each loop or path are then combined through the topology equation (3) to obtain the equivalent w-function, $w_E(s)$, between the two nodes of interest for a given path A as follows:

$$w_E(s) = \frac{w_A(s)\left[1 + \sum_{i=1}^{\infty} (-1)^i \sum_{k=1}^{n_i} w_{L_k}^{(i)}(s)\right]}{\left[1 + \sum_{j=1}^{\infty} (-1)^j \sum_{v=1}^{n_j} w_{L_v}^{(j)}(s)\right]} = \frac{A(s)\ B(s)}{D(s)} = \frac{N(s)}{D(s)}$$

where $w_A(s) \equiv$ product of the values of all branches in the path considered;

$w_{L_k}^{(i)}(s)$ = product of the values of i disjoint loops having no nodes in common with path A;

$n_i$ = the number of loops composed of i disjoint loops;

$w_{L_v}^{(j)}(s)$ = product of the values of any j disjoint loops;

$n_j$ = the number of loops composed of j disjoint loops;

and A(s), B(s), D(s) and N(s) are direct substitutions. If there is more than one path, then the w-functions associated with each path would be summed. For convenience, consider the one path case. The output statistics can be computed from the following equations:

$$p_E = w_E(o)$$

$$\mu_{1E} = \frac{1}{w_E(o)} \left. \frac{dw_E^2(s)}{ds} \right|_{s=o} = \frac{1}{w_E(o)} \left[ \frac{D(s)\,\frac{dN(s)}{ds} - N(s)\,\frac{dD(s)}{ds}}{D^2(s)} \right]_{s=o}$$

$$\mu_{2E} = \frac{1}{w_E(o)} \left. \frac{dw_E^2(s)}{ds^2} \right|_{s=o}$$

$$= \frac{1}{w_E(o)} \left\{ \frac{D(s)\left[D(s)\,\frac{d^2N(s)}{ds} - N(s)\,\frac{d^2D(s)}{ds^2}\right] - 2\,\frac{dD(s)}{ds}\left[D(s)\,\frac{dN(s)}{ds} - N(s)\,\frac{dD}{ds}\right]}{D^3(s)} \right\}_{s=o}$$

and $\sigma_E^2 = \mu_{2E} - \mu_{1E}^2$ .

In the above equations the values of $\frac{dN(s)}{ds}$ , $\frac{d^2N(s)}{ds^2}$ , etc., evaluated at s=o, are obtained from the previously compiled values of $\mu_{1L}$ and $\mu_{2L}$ as follows:

Since $N(s) = A(s)\,B(s)$,

we have

$$\frac{dN(s)}{ds} = \frac{dA(s)}{ds}\,B(s) + A(s)\,\frac{dB(s)}{ds}$$

$$\left.\frac{dN(s)}{ds}\right|_{s=o} = \frac{dA(o)}{ds}\,B(o) + A(o)\,\frac{dB(o)}{ds} \quad .$$

Now

$$A(o) = w_A(o),$$

$$\frac{dA(o)}{ds} = \mu_{1A}\,w_A(o),$$

and $$B(o) = 1 + \sum_{i=1}^{\infty} (-1)^i \sum_{k=1}^{ni} w_{L_k}^{(i)}(o)$$

$$\frac{dB(s)}{ds}\bigg|_{s=o} = \sum_{i=1}^{\infty} (-1)^i \sum_{k=1}^{n_i} \frac{dw_{L_k}^{(i)}(s)}{ds} \quad s=o$$

$$= \sum_{i=1}^{\infty} (-1)^i \sum_{k=1}^{\mu_i} \mu_{1L_k} w_{L_k}^{(i)}(o).$$

Combining terms yields

$$\frac{dN(s)}{ds}\bigg|_{s=o} = \mu_{1A}\, w_A(o) \left[1 + \sum_{i=1}^{\infty} (-1)^i \sum_{k=1}^{n_i} w_{L_k}^{(i)}(o)\right]$$

$$+ w_A(o) \left[\sum_{i=1}^{\infty} (-1)^i \sum_{k=1}^{n_i} \mu_{1L_k}^{(i)}\, w_{L_k}^{(i)}(o)\right]$$

$$= w_A(o) \left[\mu_{1A} + \sum_{i=1}^{\infty} (-1)^i \sum_{k=1}^{n_i} w_{L_k}^{(i)}(o)\, (\mu_{1A} + \mu_{1L_k}^{(i)})\right]$$

<u>Organization of the Report</u>

The next section presents a description of the overall program operation using a set of general flow charts. The third section defines the program variables and details the subprograms. For the main program and each of the eight subroutines, the following will be presented: (1) a description of what the program segment does, (2) a detailed flow chart, and (3) a program listing. The fourth section presents a discussion of the arrays of the program. This discussion enables a user to determine the size of the arrays and to change them depending on his machine storage availability and the particular network under study.

## Description of Overall Program Operation

The GERT EXCLUSIVE-OR program is composed of a main program and eight subroutines. The eight subroutine names in the basic order in which they appear in the program are: INPUT, IL, IP, LV, CL, CLP, PV, and PRP. The main program is primarily a subroutine-calling program and directly calls five of the eight subroutines. The other three--CL, CLP, and PRP--are called by other subroutines.

A general flow chart of the program is shown in Fig. 1. The following description is given to amplify a few of the operation descriptions of the flow chart. The names shown in parentheses are the names of the subroutines where the described actions take place. The starting point for the GERT program is the main program, but the only program exit point is in subroutine INPUT. An EXIT occurs when a negative value is obtain in field 1 of a data card. The first error message indicated is one that says that a bad input code was detected in the input data. The entire input network will be read in,(so that other input errors may be detected), but due to the error, the network will not be analyzed. The next network is then considered. The second error message is printed if the number of entries in the dimensioned variable LOOP(·) exceeds the size of LOOP. The remainder of the general flow chart is self-explanatory.

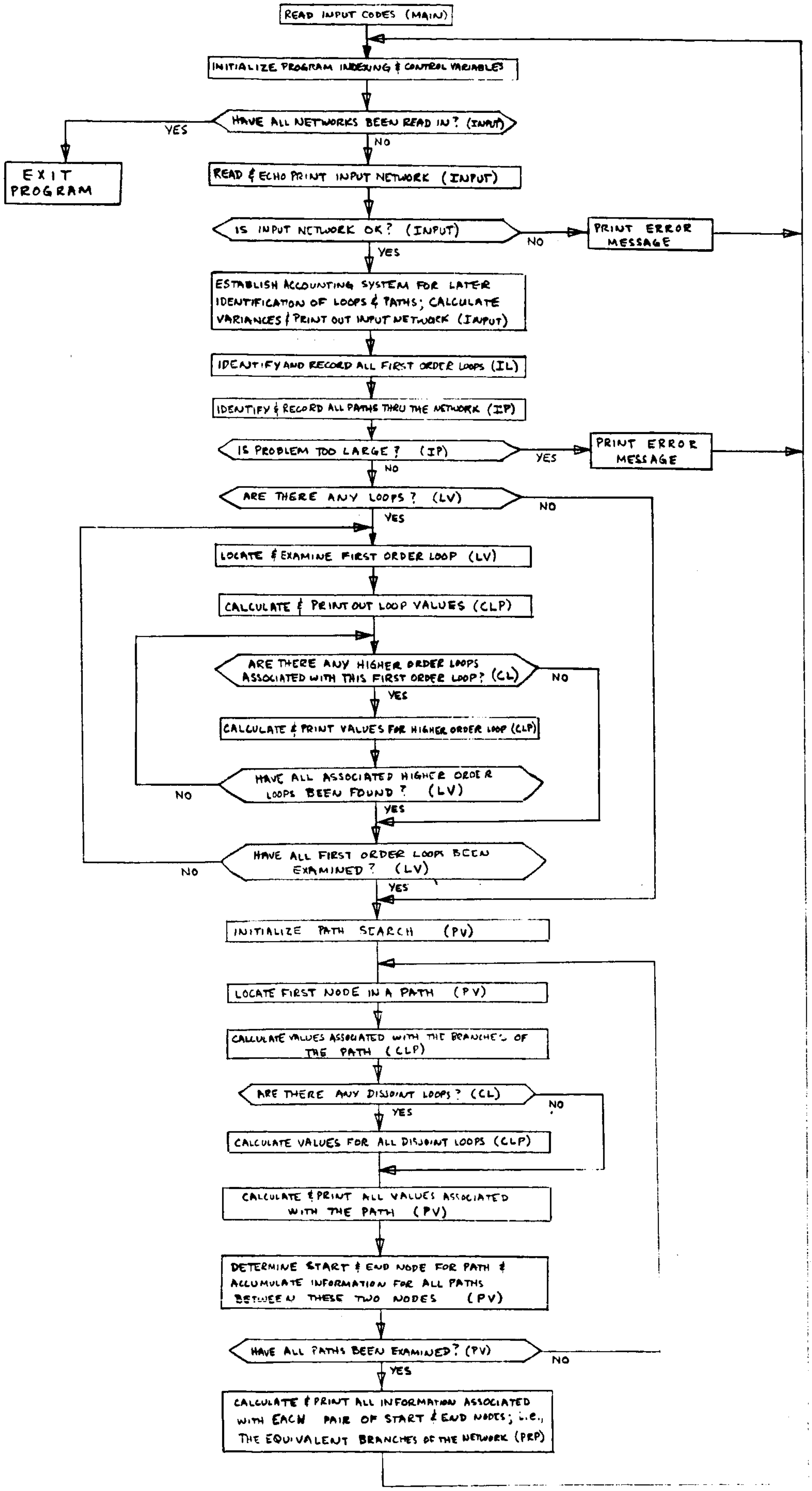



Fig. 1 General Flow Chart of GERT EXCLUSIVE-OR Program

## Detailed Description of the Program Components

The detailed description of the program will be given by first defining the program variables followed by a detailed flow chart, program listing, and description for the main program and each of the eight subroutines.

### Definition of the Program Variables

Some of the variables have more than one meaning. Where this situation exists, all definitions will be given and the subroutine where each definition applies will be named. In some cases rather than giving several definitions for a given variable or set of variables, only the general function of the variable will be described. For dimensioned variables, the number shown in parentheses indicates the array size of the variable for both the GE 225 and the IBM 1130 versions.

B(8) = Probabilities and times on the input cards (INPUT); also used to calculate loop or path values in subroutine CLP where B(1) is used for the probability calculation and B(2), B(3), and B(4) are used to calculate mean and variance of the traversal time.

D = Product of probability and time for discrete distribution (INPUT); later set to zero in subroutine LV to indicate that we are looking for loops when we use subroutine CLP; just prior to exit from subroutine LV, it is set equal to the algebraic sum of the probabilities of all loops (or = 1 if there are no loops). Since D is then nonzero, it indicates that we no longer need the loop printout on subsequent entries into subroutine CLP.

D1 = Algebraic sum of the 1st moments of the times to traverse all loops (LV).

D2 = Algebraic sum of the 2nd moments of the times to traverse all loops (LV).

DEL = Value of the deletion probability for higher order loops; for DEL = 0, no loops are deleted; for DEL > 0, all loops whose probability of realization is $\leq$ DEL are deleted. (CLP).

F = Sum of probabilities for a branch with a discrete distribution of time (INPUT); used (in LV) to accumulate the algebraic sum of the probabilities of all loops ( = 1 if there are no loops ); used (in PV) to calculate the probability of traversal of a path.

F1 = Numerator for calculation of the first moment of the time to traverse a branch having a discrete time distribution (INPUT); used (in LV) to accumulate the algebraic sum of the first moments of the times to traverse all loops (= 0 if there are no loops); used (in PV) to calculate the first moment of the time to traverse a path.

F2 = Numerator for calculation of the second moment of the time to traverse a branch having a discrete time distribution (INPUT); used (in LV) to accumulate the algebraic sum of the second moments of the times to traverse all loops (= 0 if there are no loops); used (in PV) to calculate the second moment of the time to traverse a path.

F3(50), F4(50), F5(50), F6(50) = Used to accumulate probabilities and times for loop and path calculations (CLP).

G(50), G1(50), G2(50) = These variables are used to accumulate the probability and the values needed to compute the mean and variance of the time for the equivalent branches of the network (PV).

GP = Probability of realization of a path through the network--as such it is also used as the denominator for the calculation of the first and second moments of time to traverse a path (PV); later used for the probability of realizing a given equivalent branch of the network (PRP).

GP1 = Numerator for the calculation of the first moment of time to traverse a path through the network (PV).

GP2 = Numerator for the calculation of the second moment of time to traverse a path through the network (PV).

GT = Sum of the probabilities for all equivalent network branches emanating from a given source node (PRP).

I1 through I9 =

Used throughout the program as indexing variables to aid in the identification of loops and paths and for calculating the loop and path values. In addition, the following two special uses should be noted.

I8 = Used in INPUT to indicate whether an input code error has been detected so that appropriate action may be taken upon return to the main program. $I8 \leq 0$ indicates input is all right; $I8 > 0$ indicates that a bad code was detected.

I9 = Used in IP to indicate whether the problem has become too large. $I9 \leq 0$ indicates that the problem is all right while $I9 > 0$ indicates that the problem is too large.

II = Used only as the index for a DO loop (INPUT).

J(9) = An alphabetic array containing the distribution code letters "ABDEGNOPU" for comparison with the codes in the input data (INPUT).

JCOR = Correction option for loop deletion: if JCOR > 0, the probabilities and times for the equivalent network branches are adjusted so that the probabilities for all equivalent branches emanating from a given source node sum to one.

JJ, JK, JL, JM = Used only as index adjusters; used in INPUT, IP, LV, PV, respectively.

JND = Dummy variable used in each input network control card because the program checks the first four columns of each card for a -1 to end the computer run. JND is always equal to zero in the control card.

K(3) = An alphabetic array into which the input distribution code letters are read for comparison with the J(·) array. (INPUT).

L(100) = Used in IL to identify each node appearing in a first order loop and in IP to identify each node appearing in a path; also, in subroutines LV, CL, CLP, and CLP it is used to keep track of where to start looking for the next loop or path after finishing with the loop or path being considered.

LE(100) = End node for a branch in the input network (INPUT).

LLO = Used in IL to save the last index number for LOOP(·); this index number will be one larger than the number required for saving the node numbers for nodes appearing in first order loops and will tell subroutine IP where to put the first path node.

LOOP(1000) = An array used to save the node numbers of all nodes appearing in first order loops or network paths. A LOOP(·) value of zero separates each loop and each path.

LPO = Used in IP to save the last index number for LOOP(·); this index will be one larger than the index for the last path.

LS(100) = Start node for a branch in the input network (INPUT).

MON = Part of problem heading information: month of the year (INPUT, PRP).

N(100) = Part of the accounting system for locating branches in the input network (INPUT). N(i) corresponds to node i; the value of N(i) tells which subscript of NL(·) to begin the search for branches containing node i.

N1 = Used in INPUT for reading in the start node for an input branch; later becomes the start node of a particular path through the network (PV); finally is used to print out the start node of an equivalent branch of the network (PRP).

N2 = Used in INPUT for reading in the end node of an input branch; later becomes the end node of a particular branch through the network (PV); finally is used to print out the end node of an equivalent branch of the network (PRP).

NAME(6) = An alphabetic array used to read in and print out the program user's name.

NCRD = Applies to the IBM 1130 and the CDC 3400 FORTRAN IV versions only and is the number of the card reader.

NDY = Day portion of date (INPUT, PRP).

NE(50) = End node for an equivalent branch of the network (PV,PRP).

NJOB = User's identification number for a particular problem (INPUT, PRP).

NL(200) = Part of the accounting system for locating branches in the input network (INPUT). The NL(·) value tells where to locate a particular node in the input network: it gives the subscript for LS(·) of LE(·) for node i and is positive for end nodes, LE(1), and negative for start nodes, LS(·).

NLO = The number of branches in the input network (INPUT).

NLP = Print option for loops: NLP > 0, no loop printout; NLP $\leq$ 0, loops are printed out.

NN(100) = Part of the accounting system for locating branches in the input network (INPUT). NN(i) tells how many times node i appears in the input network; if NN(i) = 1, then node i is a source or a sink node (the positive or negative sign for the associated NL(·) value tells which it is). A source or sink node may, however, appear more than once in the input network.

NNO = The largest node number appearing in the input network (INPUT).

NPP = Print option for paths: NPP > 0, no path printout, NPP $\leq$ 0, paths are printed out.

NPRT = Applies only to the FORTRAN IV versions and is the number of the printer.

NS(50) = Start node for an equivalent branch of the network (PV, PRP).

NYR = Year portion of the date (INPUT, PRP).

P(100) = Probability associated with each branch of the input network (INPUT).

T = First moment of the time to traverse a path through the network (PV); also, the first moment of the time to traverse a equivalent branch of the network (PRP).

T1(100) = First moment of the time to traverse an input branch of the network (INPUT).

T2(100) = Second moment of the time to traverse an input branch of the network (INPUT).

VT = Variance of the time to traverse an input branch of the network

(INPUT); variance of the time to traverse a path through the network (PV); and the variance of the time to traverse an equivalent branch of the network (PRP).

The Main Program

The detailed flow chart for the main program is shown in Fig. 2. At statement 100, the indexing variables are initialized and the main program calls the appropriate subroutines to read and analyze the input network. The two decision blocks represent the situations when errors have occurred and the appropriate action is to return to statement 100, re-initialize, and start on the next network. The FORTRAN statements comprising the main program are shown in Fig. 3. To facilitate conversion to other machines all input and output statements use the variables NCRD = 2 for the card reader and NPRT = 3 for the printer.

Subroutine INPUT

Subroutine INPUT is the largest of the program subroutines. INPUT initializes arrays and reads the data that describes the network. An echo check is printed out for each branch of the network. INPUT then determines the probability, mean, and variance of each branch and prints this information. INPUT also sets up an accounting system or map for locating the nodes of the network.

The flow chart for subroutine INPUT is shown in Fig. 4. In describing the activities taking place in various portions of the flow chart, Example 1 from the user's manual will be used. The network is shown in Fig. 5. The input data corresponding to the network of Fig. 5 is shown in Fig. 6. The

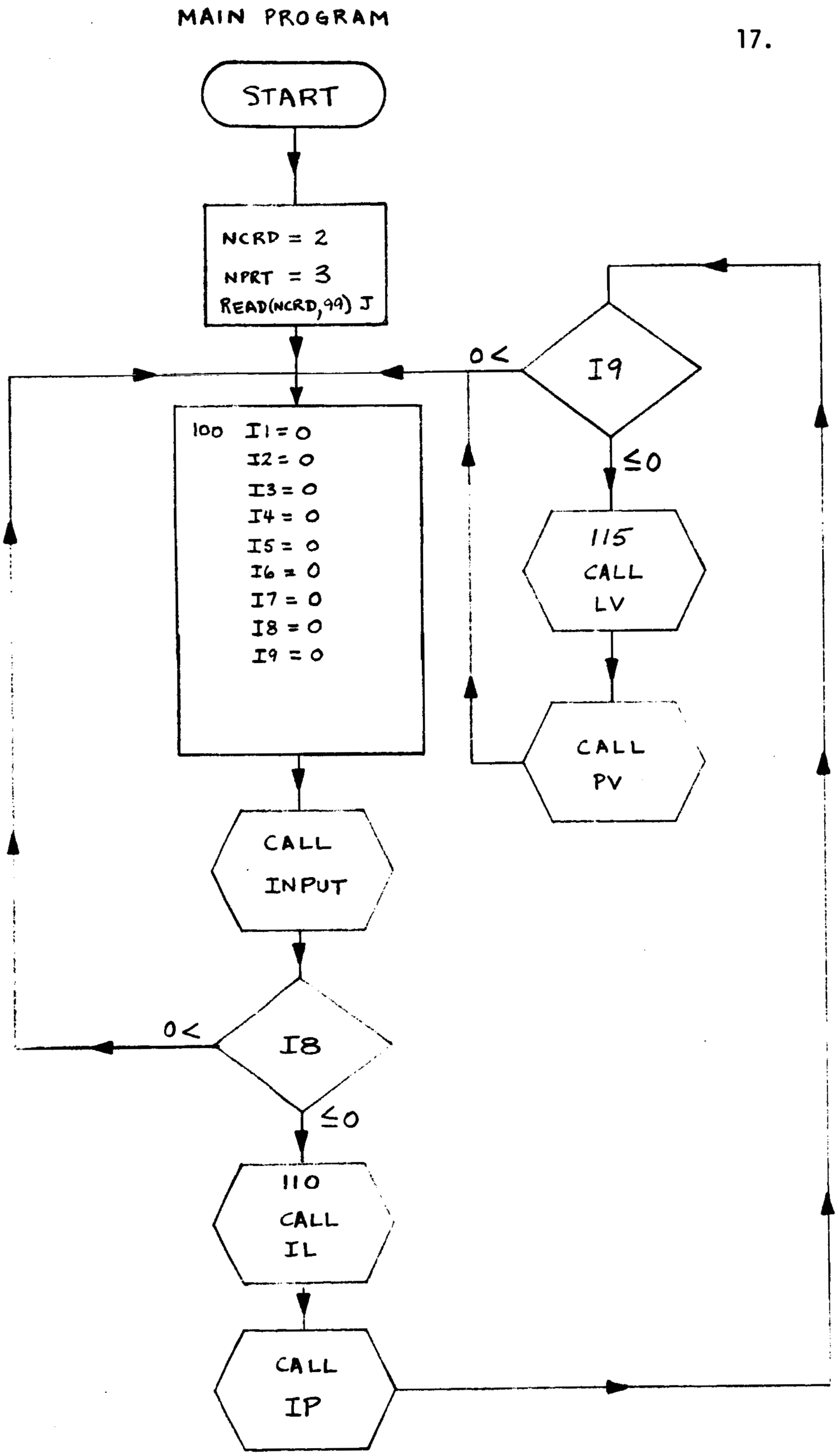



Fig. 2 Flow Chart of the Main Program

```
*IOCS(CARD,1132 PRINTER)
C
C*****GRAPHICAL EVALUATION AND REVIEW TECHNIQUE
C*****EVALUATE PROBABILITIES AND MEAN AND VARIANCE OF TIME
C*****NOTE *** FIRST DATA CARD MUST BE ABDEGNOPU CARD ***
C*****LAST DATA CARD MUST CONTAIN A -1 IN COLUMN 4
C*****LAST DATA CARD FOLLOWS BLANK OF LAST NETWORK DATA SET
C*****THIS PROGRAM HAS BEEN REVISED TO RUN ON THE IBM 1130
C*****UPDATED VERSION        **** 6-12-68 ****
C
      COMMON I1,I2,I3,I4,I5,I6,I7,I8,I9,
     1        NNO,NLO,LLO,LPO,NLP,NPP,NCRD,NPRT,JCOR,
     2F,F1,F2,F3(50),F4(50),F5(50),F6(50),P(100),
     3        D,D1,D2,N1,N2,GP,GP1,GP2,T,VT,
     4T1(100),T2(100),LS(100),LE(100),N(100),NN(100),NL(200),L(100)
      COMMON LOOP(1000),NS(50),NE(50),G(50),G1(50),G2(50),K(3),B(8),
     1J(9),NAME(6),NJOB(2),MON,NDY,NYR,DEL
   99 FORMAT(9A1)
      NCRD=2
      NPRT=3
      READ(NCRD,99) J
C
C*****INITIALIZE PROGRAM INDEXING AND CONTROL VARIABLES
C
  100 I1=0
      I2 = 0
      I3 = 0
      I4 = 0
      I5 = 0
      I6 = 0
      I7 = 0
      I8 = 0
      I9 = 0
      CALL INPUT
      IF(I8)110,110,100
  110 CALL IL
      CALL IP
      IF(I9)115,115,100
  115 CALL LV
      CALL PV
      GO TO 100
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CORE REQUIREMENTS FOR
 COMMON    3184  VARIABLES      0  PROGRAM     96

END OF COMPILATION
```

Fig. 3   FORTRAN Listing of the Main Program

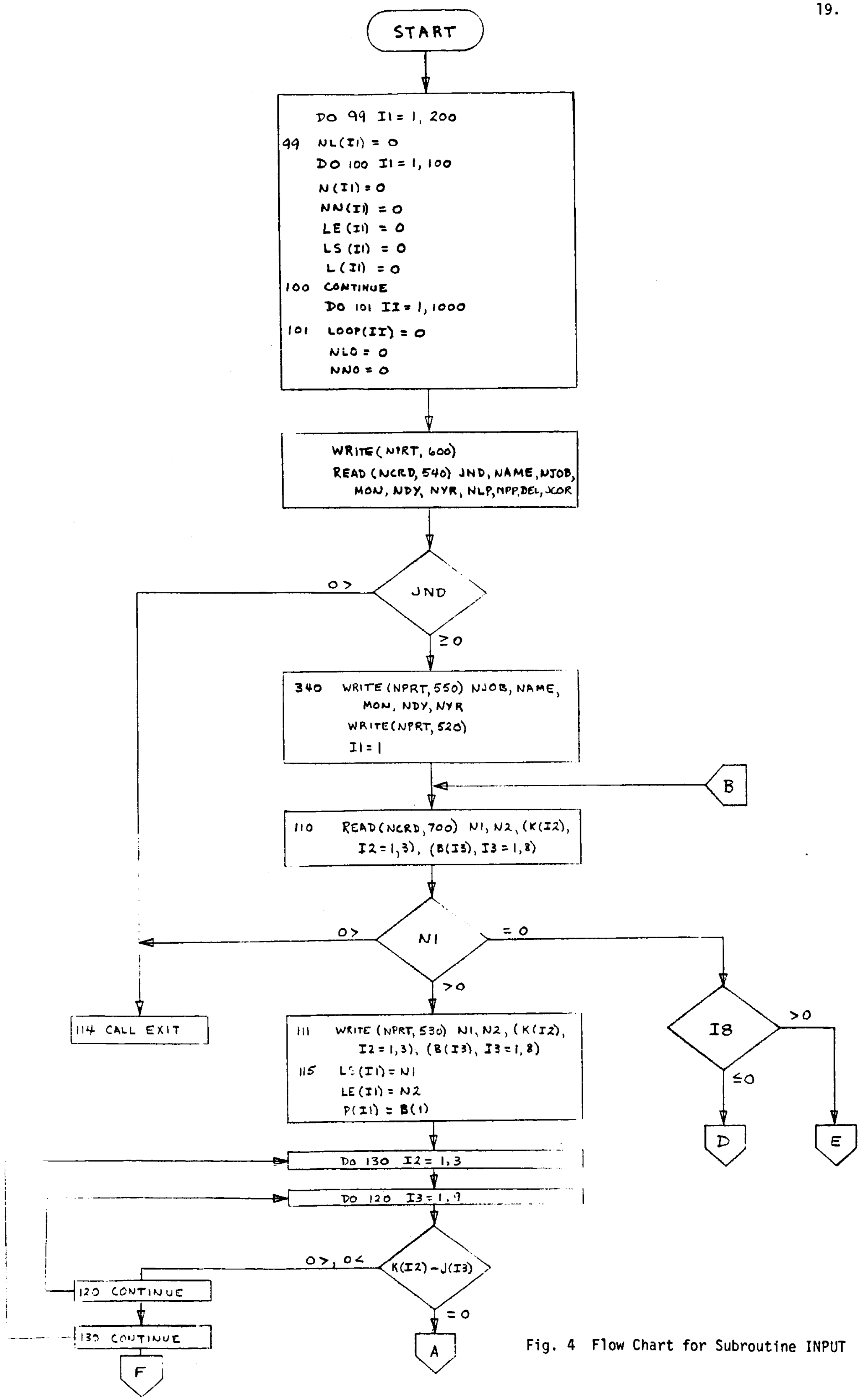



Fig. 4 Flow Chart for Subroutine INPUT

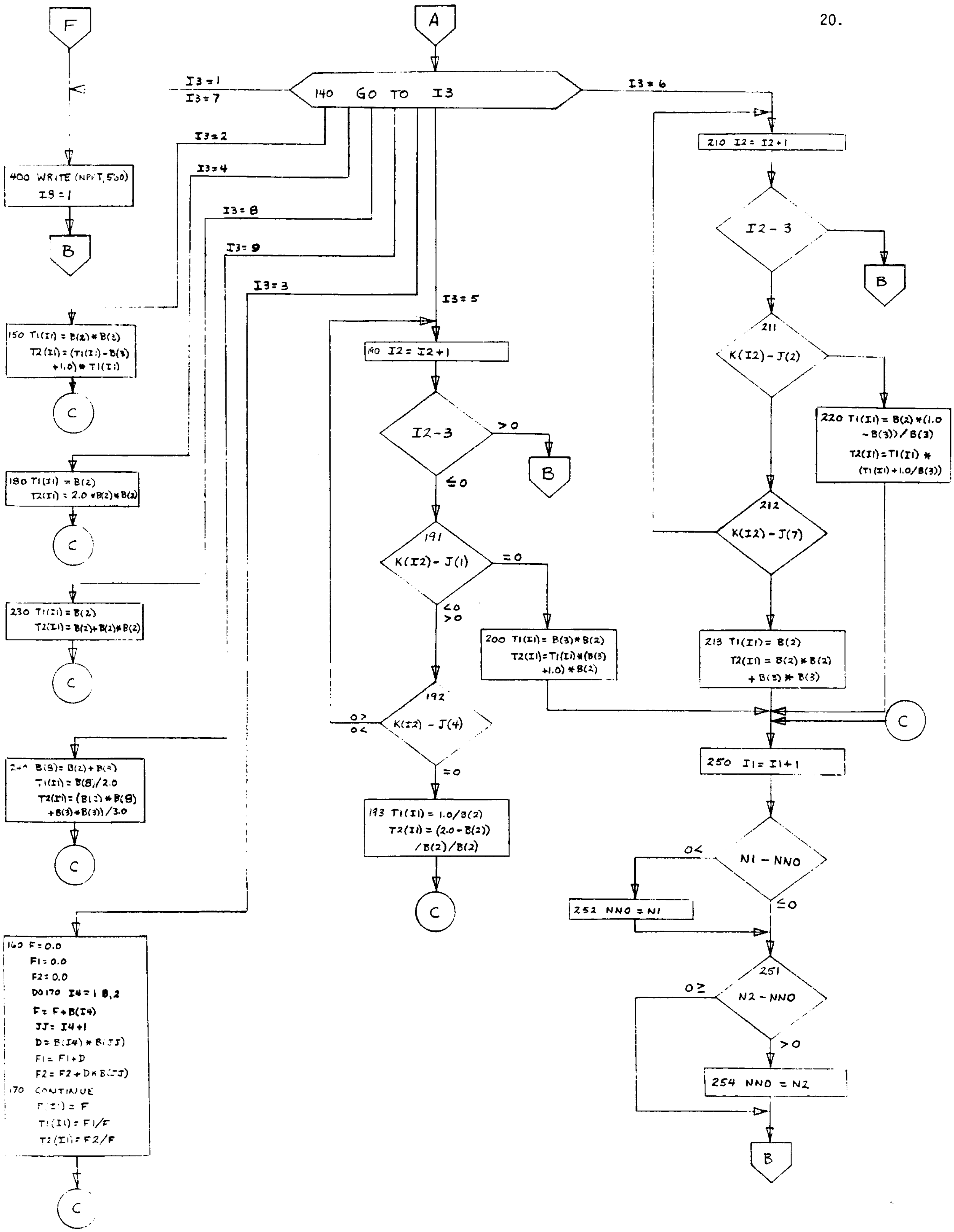



Fig. 4 (Continued)

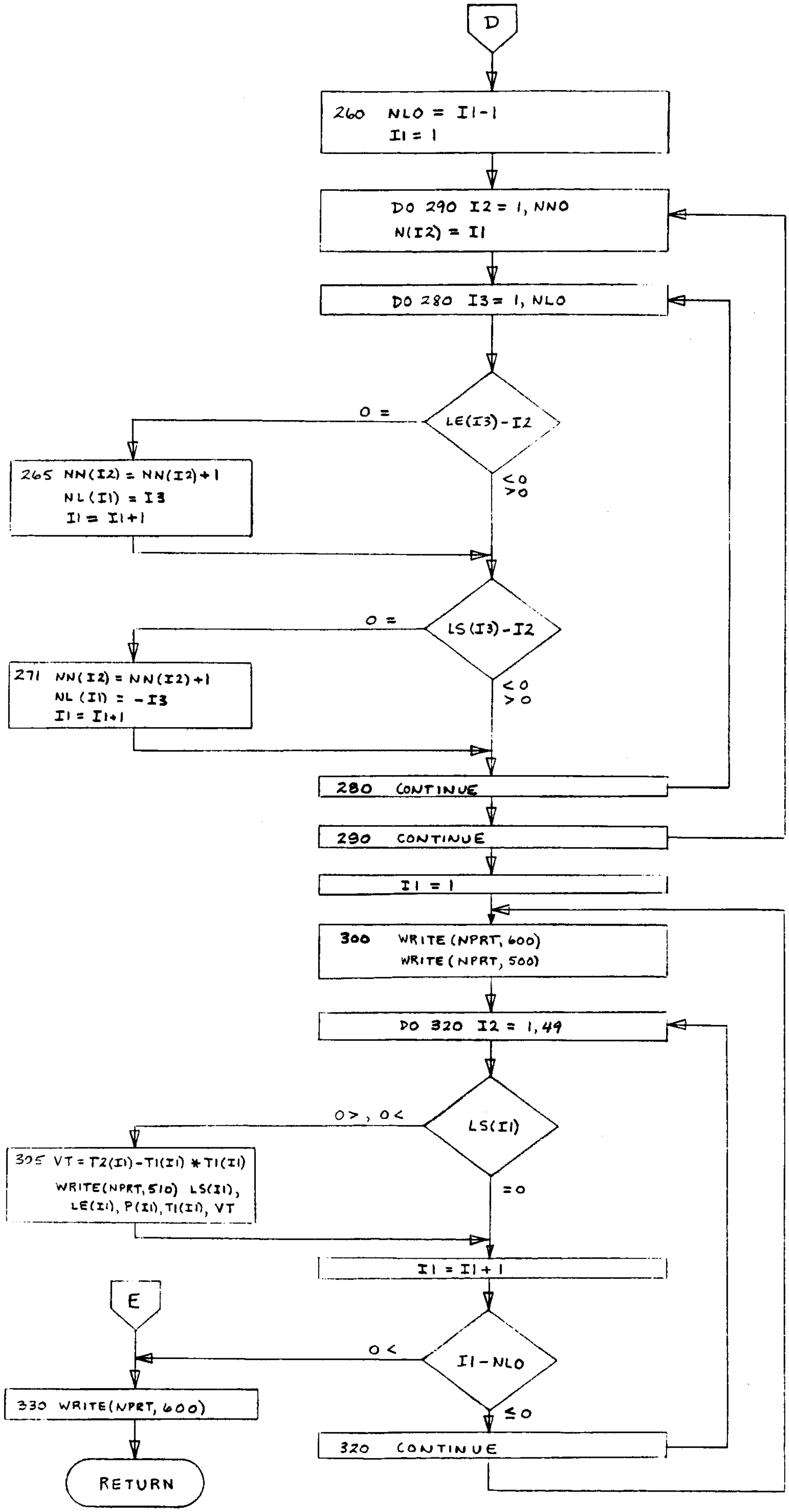



Fig. 4 (Concluded)

card numbers are included for explanatory purposes only and are not a part of the input. The "ABDEGNOPU" card is numbered 00 because it is read in by the main program. Card 0 is the heading and control card. Each branch of the network of Fig. 5 is represented by one card as shown in cards 1 through 10 in Fig. 6. An number of branches can be read in subject to the storage limitations of the machine. A blank card indicates that there are no more branches for the network and a card with a negative value in columns 1-4 indicates that there are no more networks to be analyzed.

The flow chart of Fig. 4 will now be described in terms of the data given in Fig. 6. First the important arrays and variables are zeroed. The second block shows the reading of the variables on the heading and control card (card number 0 of Fig. 6.). A check on JND is made to see if another network is to be read in since JND = 0 at this time, the heading for the echo print of the input network is printed. Card number 1 of the input network is then read in and a check is made to see if it contains a node number, a zero or a negative value. A zero would indicate that all branches have been read for the network and a negative value would indicate an illogical condition. In the latter case, the program would make a normal exit at this point.

Since card number 1 is a valid branch, it is echo printed and the start and end node for the branch are set equal to LS(1) and LE(1), respectively. The program then compares the time distribution code for the branch, K(I2), to the code contained in the program, J(I3), until it determines the distribution type of the branch. On the second page of the flow chart at point A, the distribution has been determined and transfer is made to the appropriate equations for calculation of the first and

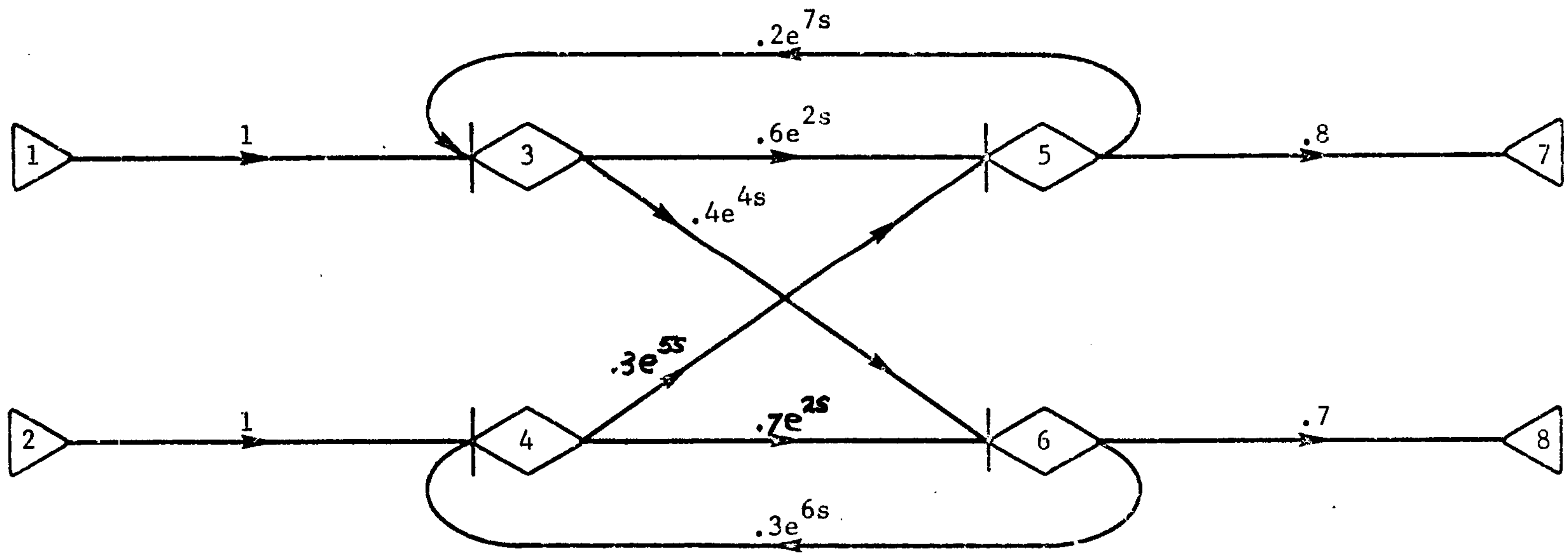



Fig. 5 GERT Network for the Example Problem

## FORTRAN CODING FORM

| | |
|---|---|
| NAME PHIL ISHMAEL | COURSE, SECTION |
| PROBLEM INPUT DATA FOR NETWORK OF FIG. 5 | DATE 5/22/68 |
| COMPILER | INSTRUCTOR |

C FOR COMMENT

| STATEMENT NUMBER (1–5) | CONT. (6) | FORTRAN STATEMENT (7–72) | Card identification number (73–80) |
|---|---|---|---|
| ABDEG | N | OPU | 00 |
| P | H | IL ISHMAEL 1 5221968 | 0 |
| 1 | | 3 D 1.0 0.0 | 1 |
| 2 | | 4 D 1.0 0.0 | 2 |
| 3 | | 5 D 0.6 2.0 | 3 |
| 3 | | 6 D 0.4 4.0 | 4 |
| 4 | | 5 D 0.3 5.0 | 5 |
| 4 | | 6 D 0.7 2.0 | 6 |
| 5 | | 3 D 0.2 7.0 | 7 |
| 6 | | 4 D 0.3 6.0 | 8 |
| 5 | | 7 D 0.8 0.0 | 9 |
| 6 | | 8 D 0.7 0.0 | 10 |
| BLANK | | CARD | 11 |
| -1 | | | 12 |

Fig. 6 Input Data for Network of Fig. 5

second moments of the time distribution. If an input code error is discovered, transfer is made to point F of the flow chart where an error message is printed. The code variable, I8, is set to 1 to flag the main program. When I8 = 1, the complete network is read in and echo printed but problem execution is terminated.

Since card number 1 has a valid input code, transfer is made to statement 160 (when I3 = 3) to make the proper calculations for the discrete distribution of time. After the calculations are completed, the program transfers to statement 250 (point C on the second page of the flow chart) where the largest node number in the input network is determined. Another card is then read in and the process is repeated until a blank card (card number 11) is read in. The echo print of the input network is then complete and appears as shown in Fig. 7.

When card number 11 is read in, the code, I8, is checked to see whether all input distribution codes were acceptable. Since they were for this input network, transfer is made to point D on the third page of the flow chart. The portion of the flow chart from the box containing statement 260 to the one containing statement 290 establishes the accounting system or map that is used later to locate any node in the network. The values established for the data given in Fig. 6 are shown in Table 1. The subscripts on the variables N(i) and NN(i) correspond to the node numbers in the network. The value of N(i) states the cell number in array NL(·) where predecessor nodes and successor nodes of node i can be determined. The value of NN(i) is the total number of predecessor and successor nodes to = node i. The values of NL(j), j = N(i), N(i) + 1, ..., N(i) + NN(i) - 1, specify the card number of the input network. If NL(j) is negative, node i is a start node;otherwise,it is an end node. Since the subscripts of LS(·) and LE(·) are card numbers, predecessor and successor node values can be obtained by use of NL(·), LS(·) and LE(·). For example, for node 4 the card numbers on which node 4 occurred are stored in NL(j), j = N(4), ..., N(4) + NN(4) - 1 or j = 7, . . ., 10. Since NL(7) = 2, node 4 is an end node on card 2. Since LS(2) = 2 there is a branch from node 2 to node 4. Since NL(8) = -5, there is a branch from node 4 to node LE(5) = 5.

The last portion of the flow chart calculates the variance for each branch of the input network and prints out the input network as shown in Fig. 8. The FORTRAN statements comprising subroutine INPUT are shown in Fig. 9. The comment cards included in the listing should aid in relating the FORTRAN listing to the flow chart and discussion.

Subroutine IL

Subroutine IL is called by the main program to identify and record all first order loops. It uses the accounting system or map established by subroutine INPUT for locating the network nodes. It checks to see whether a given node number appears in the input network more than once--if so, the associated branches are checked for a series of branches that lead back to the node number where the search began. A first order loop is identified as such a series, and the nodes involved in the loop are recorded. The program continues to check branches until all loops related to a particular "first node in a loop" are located. The program then continues through the input network until all nodes in the network have been considered as the first node of a loop.

The flow chart for subroutine IL is shown in Fig. 10. In order to describe the activities represented by the flow chart, the simple first

Table 1 Variables Used to Define a Map for Locating of Nodes of the Network

| | | |
|---|---|---|
| N(1) = 1 | NN(1) = 1 | NL(1) = -1 |
| N(2) = 2 | NN(2) = 1 | NL(2) = -2 |
| N(3) = 3 | NN(3) = 4 | NL(3) = 1; NL(4) = -3; NL(5) = -4; NL(6) = 7 |
| N(4) = 7 | NN(4) = 4 | NL(7) = 2; NL(8) = -5; NL(9) = -6; NL(10) = 8 |
| N(5) - 11 | NN(5) = 4 | NL(11) = 3; NL(12) = 5; NL(13) = -7; NL(14) = -9 |
| N(6) = 15 | NN(6) = 4 | NL(15) = 4; NL(16) = 6; NL(17) = -8; NL(18) = -10 |
| N(7) = 19 | NN(7) = 1 | NL(19) = 9 |
| N(8) = 20 | NN(8) = 1 | NL(20) = 10 |

| n, card number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LS(n) | 1 | 2 | 3 | 3 | 4 | 4 | 5 | 6 | 5 | 6 | 0 |
| LE(n) | 3 | 4 | 5 | 6 | 5 | 6 | 3 | 4 | 7 | 8 | 0 |

```
RT PROBLEM NO.   1  BY  PHIL ISHMAEL DATE  5/ 22/ 1968

PUT NETWORK
 1     3 D      1.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000
 2     4 D      1.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000
 3     5 D      0.600    2.000    0.000    0.000    0.000    0.000    0.000    0.000
 3     6 D      0.400    4.000    0.000    0.000    0.000    0.000    0.000    0.000
 4     5 D      0.300    5.000    0.000    0.000    0.000    0.000    0.000    0.000
 4     6 D      0.700    2.000    0.000    0.000    0.000    0.000    0.000    0.000
 5     3 D      0.200    7.000    0.000    0.000    0.000    0.000    0.000    0.000
 6     4 D      0.300    6.000    0.000    0.000    0.000    0.000    0.000    0.000
 5     7 D      0.800    0.000    0.000    0.000    0.000    0.000    0.000    0.000
 6     8 D      0.700    0.000    0.000    0.000    0.000    0.000    0.000    0.000
```

Fig. 7 Echo check of the Input Network of Table 1

INPUT NETWORK

NODES AND PROBABILITY OF SELECTION WITH
MEAN AND VARIANCE OF TIME FOR EACH LINK

| FROM | TO | PROB | MEAN | VAR |
|---|---|---|---|---|
| 1 | 3 | 1.000 | 0.000 | 0.000 |
| 2 | 4 | 1.000 | 0.000 | 0.000 |
| 3 | 5 | 0.600 | 2.000 | 0.000 |
| 3 | 6 | 0.400 | 4.000 | 0.000 |
| 4 | 5 | 0.300 | 5.000 | -0.000 |
| 4 | 6 | 0.700 | 2.000 | 0.000 |
| 5 | 3 | 0.200 | 7.000 | -0.000 |
| 6 | 4 | 0.300 | 6.000 | -0.000 |
| 5 | 7 | 0.800 | 0.000 | 0.000 |
| 6 | 8 | 0.700 | 0.000 | 0.000 |

Fig. 8 Calculation of Branch Parmaeters

```
      SUBROUTINE INPUT
C
C*****READ INPUT CARDS AND ARRANGE DATA
C
      COMMON I1,I2,I3,I4,I5,I6,I7,I8,I9,
     1       NNO,NLO,LLO,LPO,NLP,NPP,NCRD,NPRT,JCOR,
     2F,F1,F2,F3(50),F4(50),F5(50),F6(50),P(100),
     3       D,D1,D2,N1,N2,GP,GP1,GP2,T,VT,
     4T1(100),T2(100),LS(100),LE(100),N(100),NN(100),NL(200),L(100)
      COMMON LOOP(1000),NS(50),NE(50),G(50),G1(50),G2(50),K(3),B(8),
     1J(9),NAME(6),NJOB(2),MON,NDY,NYR,DEL
C
C*****B =BIN=BINOMIAL         P,N,1-Q    N(1-Q)           N(1-Q)(N+Q-NQ)
C*****D =DIS=DISCRETE         P,T,P,T    (PT+PT)/(P+P)    (PT*T+PT*T)/(P+P)
C*****E =EXP=EXPONENTIAL      P,A        1/A              2/A/A
C*****GA=GAM=GAMMA            P,A,B      B/A              B(B+1)/A/A
C*****GE=GEO=GEOMETRIC        P,1-Q      1/(1-Q)          (1+Q)/(1-Q)/(1-Q)
C*****NB=NB =NEG. BINOM.      P,R,1-Q    RQ/(1-Q)         RQ(1+RQ)/(1-Q)/(1-Q)
C*****NO=NOR=NORMAL           P,M,S      M                M*M+S*S
C*****P =POI=POISSON          P,L        L*L              L(1+L)
C*****U =UNI=UNIFORM          P,A,B      (A+B)/2          (A*A+A*B+B*B)/3
C
  700 FORMAT (I4,1X,I4,1X,3A1,8F7.3)
  600 FORMAT (1H1)
  500 FORMAT (10X,13HINPUT NETWORK//
     1         1X,39HNODES AND PROBABILITY OF SELECTION WITH/
     2         1X,39HMEAN AND VARIANCE OF TIME FOR EACH LINK//
     3         5X,4HFROM,3X,2HTO,5X,4HPROB,5X,4HMEAN,5X,3HVAR)
  510 FORMAT (5X,I4,1X,I4,2X,F8.3,1X,F8.3,1X,F8.3)
  520 FORMAT (1X,13HINPUT NETWORK)
  530 FORMAT (1X,I4,1X,I4,1X,3A1,8F8.3)
  540 FORMAT(I4,6A2,2A2,2I2,I4,2I2,F10.8,I2)
  550 FORMAT(18H GERT PROBLEM NO. ,2A2,6H  BY  ,6A2,6H  DATE,I3,1H/,
     1I3,1H/,I5//)
  560 FORMAT(/20X,28HBAD INPUT CODE IN ABOVE CARD/)
      DO 99 I1=1,200
   99 NL(I1) = 0
      DO 100 I1=1,100
      N(I1)=0
      NN(I1)=0
      LE(I1)=0
      LS(I1)=0
      L(I1)=0
  100 CONTINUE
      DO 101 II=1,1000
  101 LOOP(II)=0
      NLO=0
      NNO=0
      WRITE(NPRT,600)
C
C*****READ INPUT NETWORK HEADING CARD
C
      READ(NCRD,540) JND,NAME,NJOB,MON,NDY,NYR,NLP,NPP,DEL,JCOR
C
C*****HAVE ALL INPUT NETWORKS BEEN READ IN
C
      IF(JND)114,340,340
C
C*****PRINT HEADING FOR ECHO PRINT OF THE NETWORK
C
  340 WRITE(NPRT,550) NJOB,NAME,MON,NDY,NYR
      WRITE(NPRT,520)
      I1=1
C
C*****READ A CARD OF THE INPUT NETWORK
C
  110 READ(NCRD,700) N1,N2,(K(I2),I2=1,3),(B(I3),I3=1,8)
C
C*****IS THIS THE LAST CARD OF THE INPUT NETWORK
C
      IF(N1)114,410,111
  410 IF(I8)260,260,330
```

Fig. 9 FORTRAN Listing of Subroutine INPUT

```
C
C*****IS NOT LAST CARD***ECHO PRINT INPUT CARD
C
  111 WRITE(NPRT,530) N1,N2,(K(I2),I2=1,3),(B(I3),I3=1,8)
      LS(I1)=N1
      LE(I1)=N2
      P(I1)=B(1)
C
C*****CHECK FOR TIME DISTRIBUTION AND PERFORM CALCULATIONS
C
      DO 130 I2=1,3
      DO 120 I3=1,9
      IF(K(I2)-J(I3))120,140,120
  120 CONTINUE
  130 CONTINUE
      GO TO 400
  140 GO TO (400,150,160,180,190,210,400,230,240),I3
C
C*****BAD DISTRIBUTION CODE IN INPUT CARD
C
  400 WRITE(NPRT,560)
      I8=1
      GO TO 110
C
C*****BINOMIAL DISTRIBUTION
C
  150 T1(I1)=B(2)*B(3)
      T2(I1)=(T1(I1)-B(3)+1.0)*T1(I1)
      GO TO 250
C
C*****DISCRETE DISTRIBUTION
C
  160 F=0.0
      F1=0.0
      F2=0.0
      DO 170 I4=1,8,2
      F=F+B(I4)
      JJ=I4+1
      D=B(I4)*B(JJ)
      F1=F1+D
      F2=F2+D*B(JJ)
  170 CONTINUE
      P(I1)=F
      T1(I1)=F1/F
      T2(I1)=F2/F
      GO TO 250
C
C*****EXPONENTIAL DISTRIBUTION
C
  180 T1(I1)=B(2)
      T2(I1)=2.0*B(2)*B(2)
      GO TO 250
C
C*****CHECK FOR GAMMA OR GEOMETRIC DISTRIBUTION
C
  190 I2=I2+1
      IF(I2-3)191,191,110
  191 IF(K(I2)-J(1))192,200,192
  192 IF(K(I2)-J(4))190,193,190
C
C*****GEOMETRIC DISTRIBUTION
C
  193 T1(I1)=1.0/B(2)
      T2(I1)=(2.0-B(2))/B(2)/B(2)
      GO TO 250
C
C*****GAMMA DISTRIBUTION
C
  200 T1(I1)=B(3)*B(2)
      T2(I1)=T1(I1)*(B(3)+1.0)*B(2)
      GO TO 250
C
C*****CHECK FOR NORMAL OR NEGATIVE BINOMIAL DISTRIBUTION
C
  210 I2=I2+1
      IF(I2-3)211,211,110
  211 IF(K(I2)-J(2))212,220,212
  212 IF(K(I2)-J(7))210,213,210
```

Fig. 9 FORTRAN Listing of Subroutine INPUT
(continued)

```
C
C*****NORMAL DISTRIBUTION
C
  213 T1(I1)=B(2)
      T2(I1)=B(2)*B(2)+B(3)*B(3)
      GO TO 250
C
C*****NEGATIVE BINOMIAL DISTRIBUTION
C
  220 T1(I1)=B(2)*(1.0-B(3))/B(3)
      T2(I1)=T1(I1)*(T1(I1)+1.0/B(3))
      GO TO 250
C
C*****POISSON DISTRIBUTION
C
  230 T1(I1)=B(2)
      T2(I1)=B(2)+B(2)*B(2)
      GO TO 250
C
C*****UNIFORM DISTRIBUTION
C
  240 B(8)=B(2)+B(3)
      T1(I1)=B(8)/2.0
      T2(I1)=(B(2)*B(8)+B(3)*B(3))/3.0
C
C*****FIND LARGEST NODE NUMBER IN THE INPUT NETWORK
C
  250 I1=I1+1
      IF(N1-NNO)251,251,252
  252 NNO=N1
  251 IF(N2-NNO)253,253,254
  254 NNO=N2
  253 GO TO 110
C
C*****ALL BRANCHES OF THE INPUT NETWORK HAVE BEEN READ IN
C*****SET UP ACCOUNTING SYSTEM FOR IDENTIFICATION OF LOOPS AND PATHS
C
  260 NLO=I1-1
      I1=1
      DO 290 I2=1,NNO
      N(I2)=I1
      DO 280 I3=1,NLO
      IF(LE(I3)-I2)270,265,270
  265 NN(I2)=NN(I2)+1
      NL(I1)=I3
      I1=I1+1
  270 IF(LS(I3)-I2)280,271,280
  271 NN(I2)=NN(I2)+1
      NL(I1)=-I3
      I1=I1+1
  280 CONTINUE
  290 CONTINUE
      I1=1
  300 WRITE(NPRT,600)
      WRITE(NPRT,500)
      DO 320 I2=1,49
      IF(LS(I1))305,310,305
C
C*****CALCULATE VARIANCES AND PRINT OUT INPUT NETWORK
C
  305 VT=T2(I1)-T1(I1)*T1(I1)
      WRITE(NPRT,510) LS(I1),LE(I1),P(I1),T1(I1),VT
  310 I1=I1+1
      IF(I1-NLO)320,320,330
  320 CONTINUE
      GO TO 300
  330 WRITE(NPRT,600)
      RETURN
  114 CALL EXIT
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR INPUT
 COMMON    3184  VARIABLES     10  PROGRAM   1250

END OF COMPILATION
```

Fig. 9 FORTRAN Listing of Subroutine INPUT (continued)

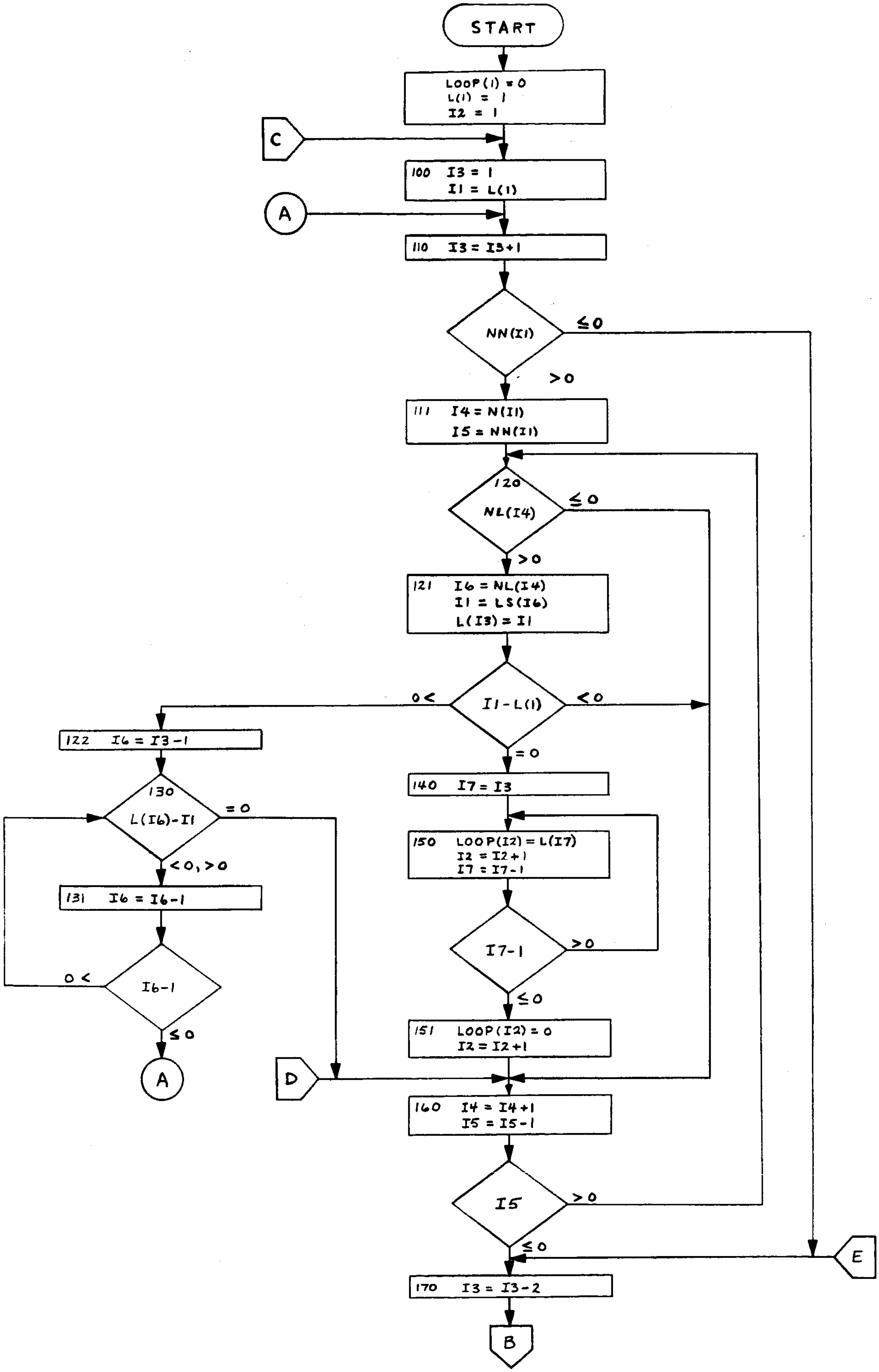



Fig. 10 Flow Chart of Subroutine JL

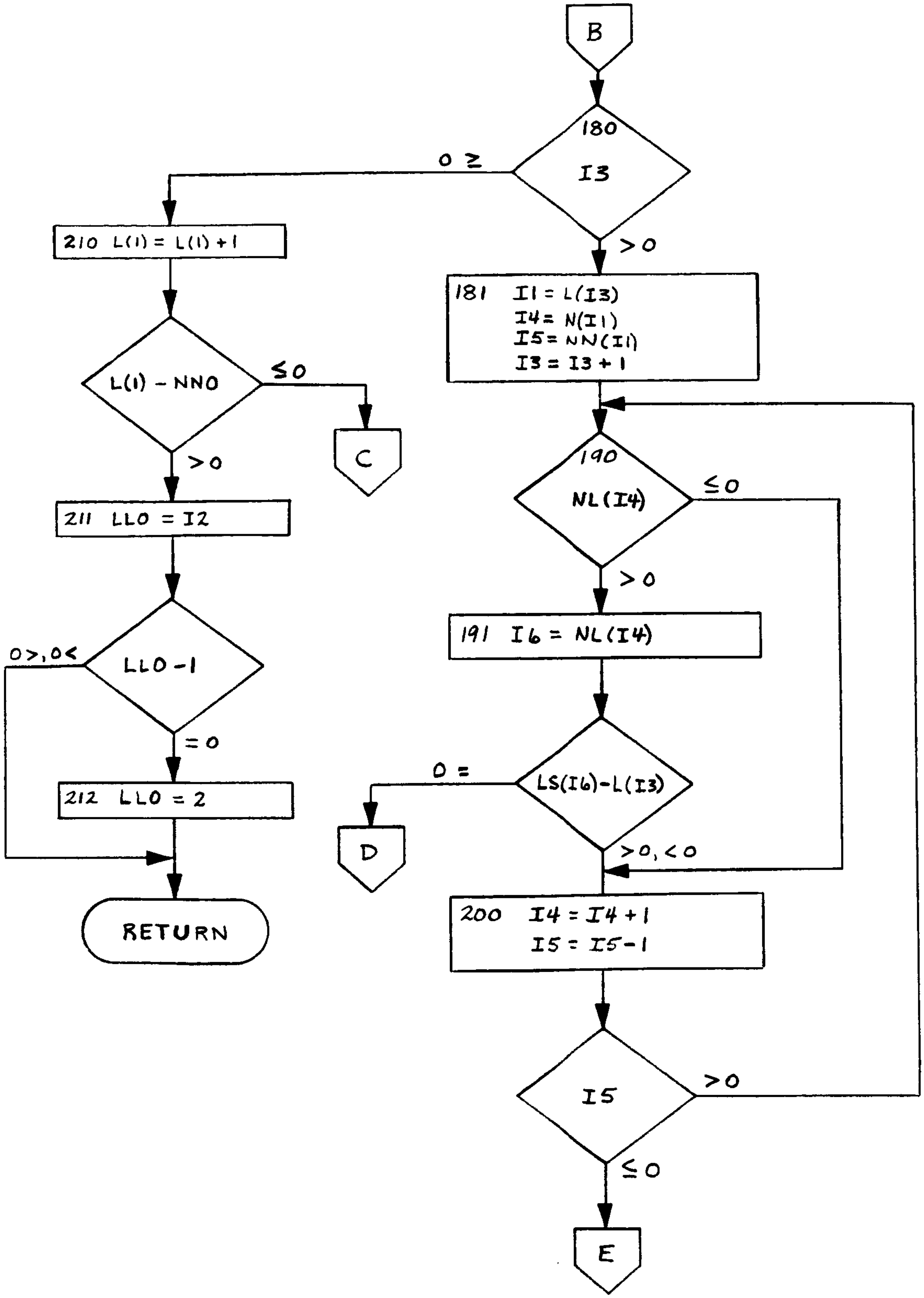



Fig. 10 (Concluded)

order loop consisting of nodes 3 and 5 of Fig. 5, will be used along with the information from Table 1 and Fig. 6. First, IL initializes three important variables: LOOP(·) will be the permanent storage location for node numbers contained in first order loops; L(·) will be the temporary storage location for node numbers that are being checked as candidates for being in loops; and I2 is the index for the subscript of LOOP(·). A glance at the first few steps in the flow chart and the values for NN(i) in Table 1 shows that node number 3 is the first candidate for being a member of a loop. Also, by the time NN(3) is reached, the temporary node storage value, L(1) = 3. The program sets up some temporary index values at statement 111 and checks for branches where node number 3 is an end node at statement 120. A glance at Fig. 6 shows that card number 1 is the first such branch so that L(2) = 1. The transfer statement following statement 121 shows that node 1 is not a satisfactory candidate for the loop so the search continues to card number 7 which is the next card in which node 3 is an end node. L(2) then is set equal to 5 which is the start node for that branch. The branches associated with node 5 are then checked until it is found that L(3) = 3 and the loop has been discovered. The nodes in the loop are recorded at statement 150 and a LOOP(·) value of zero is inserted to separate this loop from other first order loops or from paths if there are no more loops. At this time the node numbers in the loop have been recorded as: LOOP(1) = 3, LOOP(2) = 5, and LOOP(3) = 0. The process continues until all first order loops have been located. When all loops have been located, the last value of I2 is recorded as LLO at statement 211. This index number is used later as the first storage location for path nodes.

The FORTRAN statements comprising subroutine IL are shown in Fig. 11. Comment cards are included to indicate important operations.

```
      SUBROUTINE IL
C
C*****IDENTIFY LOOPS
C
      COMMON I1,I2,I3,I4,I5,I6,I7,I8,I9,
     1       NNO,NLO,LLO,LPO,NLP,NPP,NCRD,NPRT,JCOR,
     2F,F1,F2,F3(50),F4(50),F5(50),F6(50),P(100),
     3       D,D1,D2,N1,N2,GP,GP1,GP2,T,VT,
     4T1(100),T2(100),LS(100),LE(100),N(100),NN(100),NL(200),L(100)
      COMMON LOOP(1000),NS(50),NE(50),G(50),G1(50),G2(50),K(3),B(8),
     1J(9),NAME(6),NJOB(2),MON,NDY,NYR,DEL
C
C*****IDENTIFY AND RECORD ALL FIRST ORDER LOOPS
C
      LOOP(1)=0
      L(1)=1
      I2=1
  100 I3=1
      I1=L(1)
  110 I3=I3+1
      IF(NN(I1)-1)170,170,111
  111 I4=N(I1)
      I5=NN(I1)
  120 IF(NL(I4))160,160,121
  121 I6=NL(I4)
      I1=LS(I6)
      L(I3)=I1
      IF(I1-L(1))160,140,122
  122 I6=I3-1
  130 IF(L(I6)-I1)131,160,131
  131 I6=I6-1
      IF(I6-1)132,132,130
  132 GO TO 110
  140 I7=I3
C
C*****RECORD NUMBERS OF NODES CONTAINED IN THIS LOOP
C
  150 LOOP(I2)=L(I7)
      I2=I2+1
      I7=I7-1
      IF(I7-1)151,151,150
C
C*****A LOOP(I2) VALUE OF ZERO SEPARATES THE LOOPS
C
  151 LOOP(I2)=0
      I2=I2+1
  160 I4=I4+1
      I5=I5-1
      IF(I5)170,170,120
  170 I3=I3-2
      IF(I3)210,210,181
  181 I1=L(I3)
      I4=N(I1)
      I5=NN(I1)
      I3=I3+1
  190 IF(NL(I4))200,200,191
  191 I6=NL(I4)
      IF(LS(I6)-L(I3))200,160,200
  200 I4=I4+1
      I5=I5-1
      IF(I5)201,201,190
  201 GO TO 170
  210 L(1)=L(1)+1
      IF(L(1)-NNO)100,100,211
C
C*****SAVE INDEX NUMBER FOR RECORDING OF FIRST PATH NODE
C
  211 LLO=I2
      IF(LLO-1)213,212,213
  212 LLO=2
  213 RETURN
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR IL
 COMMON    3184  VARIABLES       4  PROGRAM    336

END OF COMPILATION
```

Fig. 11 FORTRAN Listing of Subroutine IL

## Subroutine IP

Subroutine IP is called by the main program to identify and record all paths through the network. The accounting system established by subroutine INPUT is used to help locate the nodes appearing in paths through the network. The subroutine starts by identifying a source node. It then proceeds from node to node through the network until it reaches a sink node without returning to any node. A path then identified as the sequence of nodes from source node to sink node with any node of the path only appearing once in the sequence.

To explain the process further, the flow chart shown in Fig. 12 will be utilized together with Fig. 5 and the accounting system of Table 1. At the start of the subroutine the indexes are initialized with I8 being set to the subscript for LOOP(·) that will be used to store the first node in the first path. This is the LLO value that was saved in subroutine IL. At statement 100 the program starts to check the first node with the check on I5 being inserted to assure that the node appears in the input network [NN(i)>0]. The set of statements from 110 to 115 are used to locate a source node which is a node which is not an end node for any branch. The node is then recorded at statement 120. The node is again checked to verify that it does indeed appear in the input network, and at statement 121, the program prepares to start from the recorded node to find the next node in the path. At statement 130, a check is made to see whether the recorded node is a start or end node. If the node is a start node, then transfer is made to point "A" on the chart where the search is made for the end node of a branch emanating from the recorded node. The check at statement 140 to

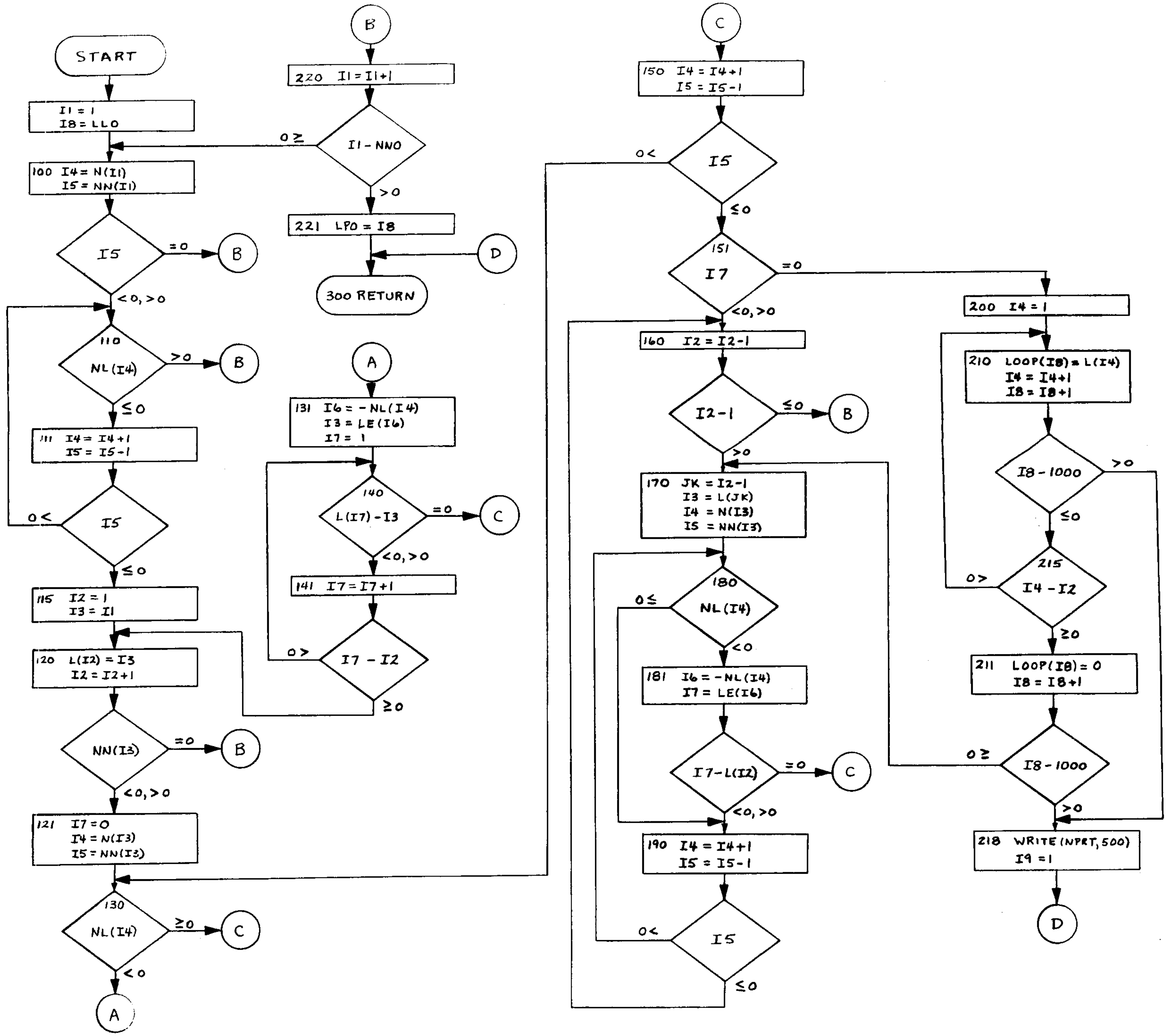



Fig. 12 Flow Chart of Subroutine IP

see if the end node being examined has already been recorded as a member of the path (the situation where L(I7) = I3). If the node has not already been recorded, then the routine returns to statement 120 where the new node is recorded and the process continues. If the check at statement 130 indicates that the node is an end node, then transfer is made to point "C" on the chart where the indexes are incremented and checked. Depending on the value of I5, the program may return to 130 or go on to 151. If it goes on to 151, the last recorded node was a sink node. If the variable I7 is 0 at statement 151, then the path nodes are recorded and a check is made to see that the subscript for LOOP(·) has not become too large (I8>1000). If the subscript is acceptable, then transfer is made to statement 170. The purpose of this portion of the subroutine is to back down the path node at a time from the sink node to the source node and search for other paths from that point to a sink node. To illustrate this process for the network of Fig. 5, the first path through the network contains nodes 1, 3, 5 and 7. The second path consists of nodes 1, 3, 6, 4, 5 and 7, and the third path contains nodes 1, 3, 6 and 8. After locating the first path, the subroutine has to back all the way to node 3 before an additional path was identified. After the second path was recorded, the subroutine only had to return to node 6 before finding the third path. On the fourth such attempt, however, no path could be found even after returning to node 1, so transfer was made to point B on the flow chart and then back to statement 100 to seek a different source node. Exit from the subroutine occurs at point B on the flow chart when the next node number to be checked is larger than any node appearing in the input network. The last index number for LOOP(·) is recorded and specifies the end of storage for paths. Since paths are loops without a closing branch, LOOP(·) can be used to store both. This eliminates a need for allocating storage between loops and paths and reduces storage requirements. Program control is then returned to the main program.

The FORTRAN statements comprising subroutine IP are shown in Fig. 13.

Subroutine LV

Subroutine LV is called by the main program to calculate the values for all loops in the network. It first checks to see whether there are any first order loops. If first order loops exist, the loop values are calculated and then all higher order loops associated with a given first order loop are identified and their values are calculated. This process is repeated until all first order loops have been examined. Program control then returns to the main program.

The flow chart for subroutine LV is shown in Fig. 14. The first box contains the initialization of the variables for calculating the probabilities and times to traverse the loops. The check on the variable LLO is to determine whether there are any first order loops (the situation where LLO>2). If there are no first order loops, the values for D, D1, and D2 are established and control is returned to the main program. If there are first order loops, then the program is directed to statement 100 which tells where to start the search for a first order loop. Subroutine CLP is then called to make the actual loop calculations and to print the loop values. Upon return from CLP, a check is made for higher order loops associated with the first order loop under consideration. The section of the subroutine from statement 110 down to the check on "I1-1" is devoted to the search for higher

* Note that LOOP(·) is used to store node values and if storage is critical several node values can be packed into a word i.e., LOOP(·) is a good candidate for packing.

```
      SUBROUTINE IP
C
C*****IDENTIFY PATHS
C
      COMMON I1,I2,I3,I4,I5,I6,I7,I8,I9,
     1       NNO,NLO,LLO,LPO,NLP,NPP,NCRD,NPRT,JCOR,
     2F,F1,F2,F3(50),F4(50),F5(50),F6(50),P(100),
     3       D,D1,D2,V1,N2,GP,GP1,GP2,T,VT,
     4T1(100),T2(100),LS(100),LE(100),N(100),NN(100),NL(200),L(100)
      COMMON LOOP(1000),NS(50),NE(50),G(50),G1(50),G2(50),K(3),B(8),
     1J(9),NAME(6),NJOB(2),MON,NDY,NYR,DEL
C
C*****IDENTIFY AND RECORD ALL PATHS THRU THE NETWORK
C
      I1=1
      I8=LLO
  100 I4=N(I1)
      I5=NN(I1)
      IF(I5)110,220,110
  110 IF(NL(I4))111,111,220
  111 I4=I4+1
      I5=I5-1
      IF(I5)115,115,110
  115 I2=1
      I3=I1
  120 L(I2)=I3
      I2=I2+1
      IF(NN(I3))121,220,121
  121 I7=0
      I4=N(I3)
      I5=NN(I3)
  130 IF(NL(I4))131,150,150
  131 I6=-NL(I4)
      I3=LE(I6)
      I7=1
  140 IF(L(I7)-I3)141,150,141
  141 I7=I7+1
      IF(I7-I2)140,142,142
  142 GO TO 120
  150 I4=I4+1
      I5=I5-1
      IF(I5)151,151,130
  151 IF(I7)160,200,160
  160 I2=I2-1
      IF(I2 - 1) 220,220,170
  170 JK=I2-1
      I3=L(JK)
      I4=N(I3)
      I5=NN(I3)
  180 IF(NL(I4))181,190,190
  181 I6=-NL(I4)
      I7=LE(I6)
      IF(I7-L(I2))190,150,190
  190 I4=I4+1
      I5=I5-1
      IF(I5)191,191,180
  191 GO TO 160
  200 I4=1
C
C*****RECORD NUMBERS OF NODES CONTAINED IN THIS PATH
C
  210 LOOP(I8)=L(I4)
      I4=I4+1
      I8=I8+1
      IF(I8-1000)215,215,218
  215 IF(I4-I2)210,211,211
C
C*****A LOOP(I8) VALUE OF ZERO SEPARATES THE PATHS
C
  211 LOOP(I8)=0
      I8=I8+1
      IF(I8-1000)170,170,218
C
C*****DIMENSION OF LOOP(I8) IS TOO LARGE, TERMINATE PROBLEM EXECUTION
C
  218 WRITE(NPRT,500)
  500 FORMAT(//20X,17HPROPLEM TOO LARGE//)
      I9=1
      GO TO 300
  220 I1=I1+1
      IF(I1-NNO)100,100,221
C
C*****SAVE THE LAST VALUE OF I8----IT HELPS LOCATE THE LAST PATH
C
  221 LPO=I8
  300 RETURN
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR IP
 COMMON    3184  VARIABLES       4  PROGRAM    392

END OF COMPILATION
```

Fig. 13 FORTRAN Listing of Subroutine IP

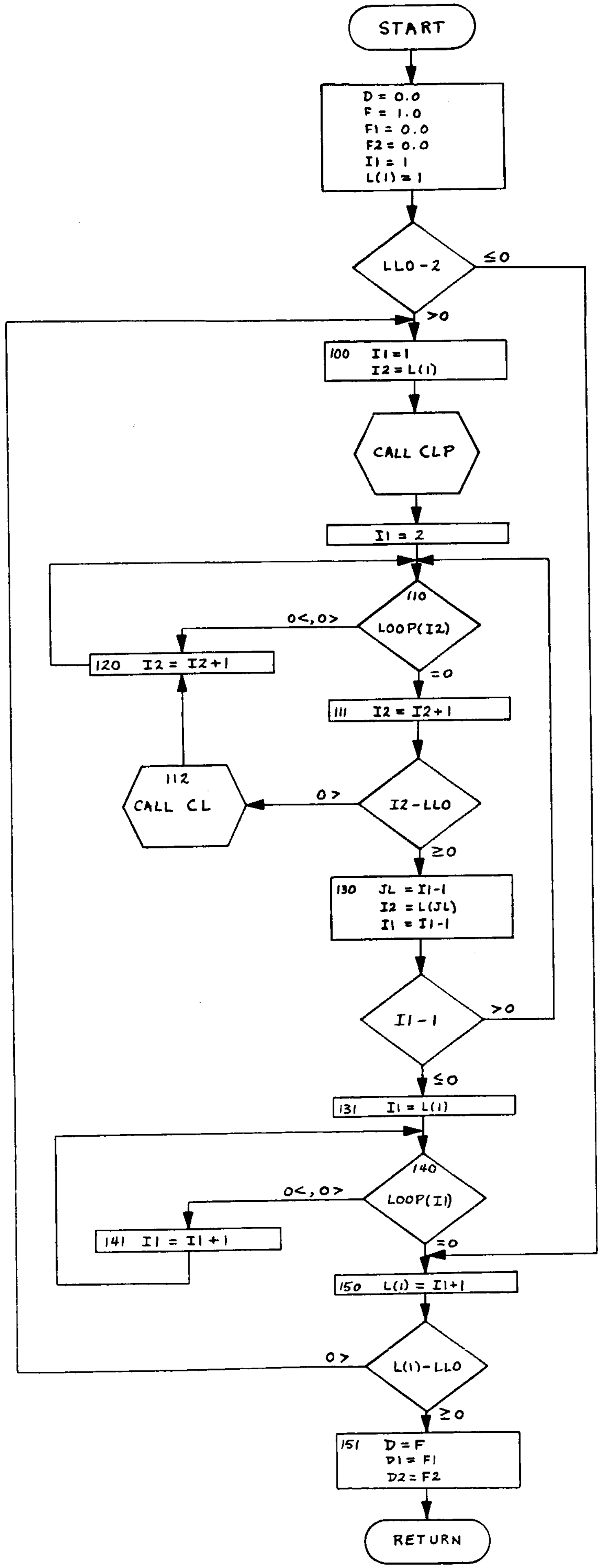



Fig. 14 Flow Chart of Subroutine LV

order loops. In this section, if additional first order loops exist, then subroutine CL is called to see if any of these additional loops are disjoint from the first order loop under consideration. If such disjoint or higher loops are found, then CL calls subroutine CLP to calculate and print out the loop values. Then return is made from CLP to CL to LV. After all such higher order loops are found, the subroutine moves on eventually to the check on "L(1) - LLO," which is a check to see whether all first order loops have been examined. If not, the program returns to statement 100 and repeats the above process. If all loops have been examined, then the values of D, D1, and D2 are saved for use in the calculation of path values and control is returned to the main program.

The FORTRAN statements comprising subroutine LV are shown in Fig. 15.

Subroutine CL

Subroutine CL is called both by subroutine LV and by subroutine PV to locate disjoint or higher order loops. When called by LV, subroutine CL checks all remaining first order loops besides the basic one considered by LV to see whether any of the remaining loops are disjoint from the basic one. A loop is disjoint if it has no nodes in common with the basic first order loop. When called by PV, subroutine CL checks all first order loops to see whether there are any that are disjoint from the path being considered. Again, a loop is disjoint from a path if it has no nodes in common with the path. If a disjoint loop is discovered, subroutine CLP is called to calculate the values associated with the loop. Subroutine CLP then returns to CL which returns to the calling subroutine. If a

```
      SUBROUTINE LV
C
C*****LOOP VALUE
C
      COMMON I1,I2,I3,I4,I5,I6,I7,I8,I9,
     1       NNO,NLO,LLO,LPO,NLP,NPP,NCRD,NPRT,JCOR,
     2F,F1,F2,F3(50),F4(50),F5(50),F6(50),P(100),
     3       D,D1,D2,N1,N2,GP,GP1,GP2,T,VT,
     4T1(100),T2(100),LS(100),LE(100),N(100),NN(100),NL(200),L(100)
      COMMON LOOP(1000),NS(50),NE(50),G(50),G1(50),G2(50),K(3),B(8),
     1J(9),NAME(6),NJOB(2),MON,NDY,NYR,DEL
C
C*****INITIALIZE VARIABLES FOR CALCULATION OF LOOP VALUES
C
      D=0.0
      F=1.0
      F1=0.0
      F2=0.0
      I1=1
      L(1)=1
C
C*****CHECK TO SEE IF THERE ARE ANY LOOPS (LLO GT 2)
C
      IF(LLO-2)150,150,100
  100 I1=1
C
C*****L(1) TELLS WHERE NEXT FIRST ORDER LOOP STARTS
C
      I2=L(1)
C
C*****LOOP CALCULATIONS AND PRINTOUT OCCUR IN SUBROUTINE CLP
C
      CALL CLP
      I1=2
  110 IF(LOOP(I2))120,111,120
  111 I2=I2+1
C
C*****HAVE ALL ASSOCIATED HIGHER ORDER LOOPS BEEN FOUND
C
      IF(I2-LLO)112,130,130
C
C*****SUBROUTINE CL CHECKS FOR DISJOINT LOOPS ASSOCIATED WITH THIS
C*****FIRST ORDER LOOP***IF HIGHER ORDER LOOPS ARE THUS FOUND,
C*****SUB. CLP IS CALLED FROM CL TO CALCULATE AND PRINT OUT THEIR VALUE
C
  112 CALL CL
  120 I2=I2+1
      GO TO 110
  130 JL=I1-1
      I2=L(JL)
      I1=I1-1
      IF(I1-1)131,131,110
  131 I1=L(1)
  140 IF(LOOP(I1))141,150,141
  141 I1=I1+1
      GO TO 140
  150 L(1)=I1+1
C
C*****HAVE ALL FIRST ORDER LOOPS BEEN EXAMINED
C
      IF(L(1)-LLO)100,151,151
C
C*****SAVE SUM OF PROBABILITIES AND TIMES FOR ALL LOOPS
C
  151 D=F
      D1=F1
      D2=F2
      RETURN
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR LV
 COMMON    3184  VARIABLES      2  PROGRAM    166

END OF COMPILATION
```

Fig. 15 FORTRAN Listing of Subroutine LV common node is found, CL returns to the calling subroutine without calling CLP.

The flow chart for subroutine CL is shown in Fig. 16. The variable I4 is used to keep track of the order of the disjoint loop. The variable I5 is the index for checking the nodes in the basic first order loop while the variable I6 is the index for checking the nodes of the remaining first order loops against the nodes in the basic one. At statement 120, the check is made to see whether a common node exists. If not, the program goes to statement 121 where it increments the index of the loop being checked to compare the next node. If LOOP(I6) = 0, then the end of the loop has been reached and the program goes to statement 122 to increment the index for the basic loop. This process continues until a common node is found or until all nodes in the basic loop have been compared against all nodes in one of the remaining loops. At that point, a disjoint loop has been found and the value of I4 is incremented at statement 123. The variable I4 is compared against I1 to see whether a loop of the desired order has been located. If so, L(I1) is set equal to the subscript required to locate the new loop and subroutine CLP is called to calculate the loop values. Upon return from CLP, the variable I1 is incremented (to indicate what order loop to seek upon the next entry into CL) and return is made to the calling subroutine.

The FORTRAN statements comprising subroutine CL are shown in Fig. 17.

<u>Subroutine CLP</u>

Subroutine CLP is called by subroutines LV, CL, and PV to calculate

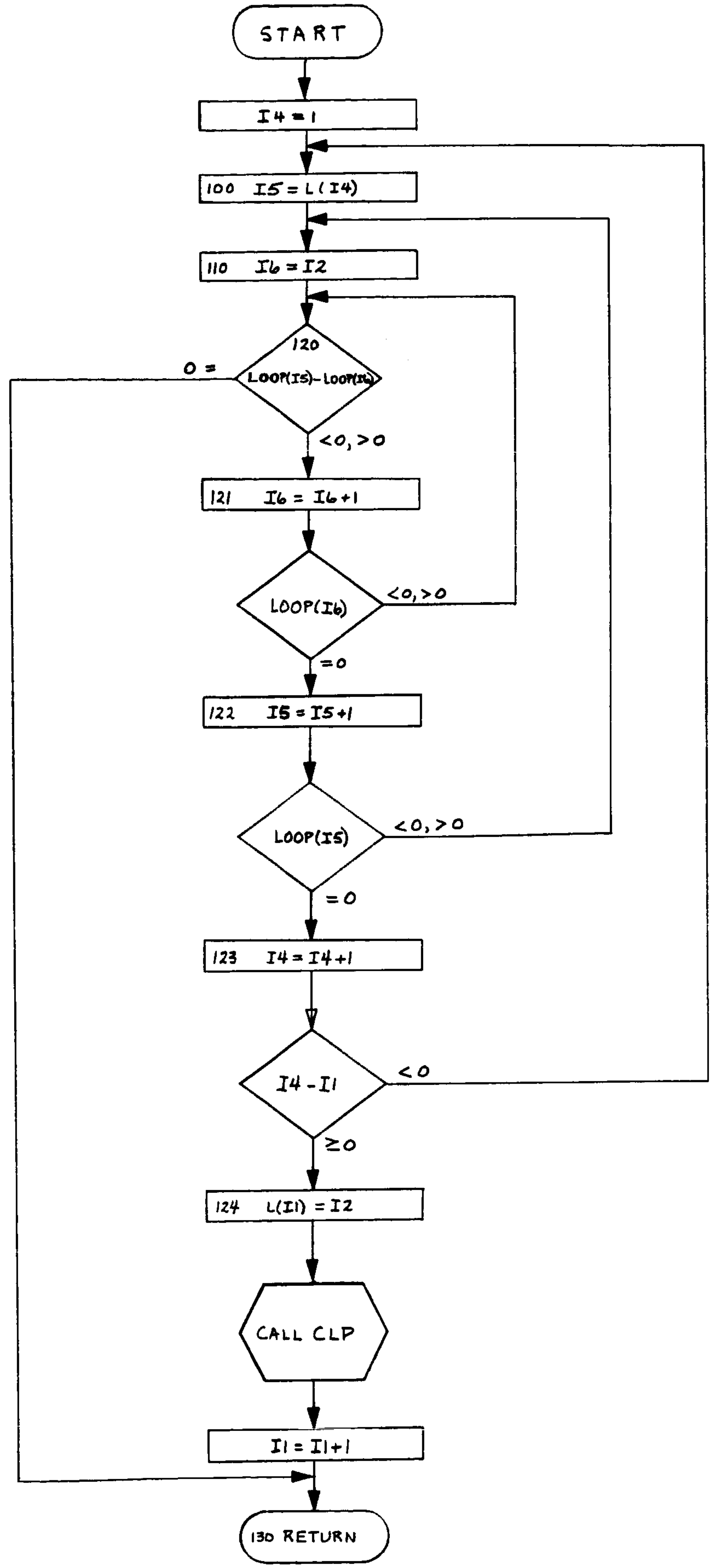



Fig. 16 Flow Chart of Subroutine CL

```
      SUBROUTINE CL
C
C*****COMPOUND LOOPS
C
      COMMON I1,I2,I3,I4,I5,I6,I7,I8,I9,
     1      NNO,NLO,LLO,LPO,NLP,NPP,NCRD,NPRT,JCOR,
     2F,F1,F2,F3(50),F4(50),F5(50),F6(50),P(100),
     3      D,D1,D2,N1,N2,GP,GP1,GP2,T,VT,
     4T1(100),T2(100),LS(100),LE(100),N(100),NN(100),NL(200),L(100)
      COMMON LOOP(1000),NS(50),NE(50),G(50),G1(50),G2(50),K(3),B(8),
     1J(9),NAME(6),NJOB(2),MON,NDY,NYR,DEL
C
C*****SUBROUTINE CL SEARCHES FOR DISJOINT LOOPS
C
      I4=1
  100 I5=L(I4)
  110 I6=I2
C
C*****IF A COMMON NODE IS FOUND, THE LOOP IS NOT DISJOINT
C
  120 IF(LOOP(I5)-LOOP(I6))121,130,121
  121 I6=I6+1
      IF(LOOP(I6))120,122,120
  122 I5=I5+1
      IF(LOOP(I5))110,123,110
  123 I4=I4+1
      IF(I4-I1)100,124,124
  124 L(I1)=I2
C
C*****WHEN A DISJOINT LOOP IS FOUND, SUB. CLP IS
C*****CALLED TO CALCULATE THE VALUES
C
      CALL CLP
      I1=I1+1
  130 RETURN
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR CL
 COMMON    3184  VARIABLES      2  PROGRAM    100

END OF COMPILATION
```

Fig. 17 FORTRAN Listing for Subroutine CL the values associated with a first order loop, higher order loops, and paths, respectively. The probability associated with a loop or path is the product of the probabilities for each branch of the loop or path. The first and second moments of time to traverse a loop or path are the times associated with the branches of the loop or path combined in the manner discussed in the first part of this report. If loop values are being calculated for the first time, then subroutine CLP prints them out unless the option to delete loop printout is exercised. Once the program has begun to calculate path values, however, the section of CLP dealing with the loop printout is no longer used even though CLP is used for the calculation of loop values for loops disjoint from the path being considered.

The flow chart of subroutine CLP is shown in Fig. 18. The calculation of loop or path values is carried out in the same way with the exception that printout of loop values occurs the first time they are calculated. The first box of the flow chart contains the initialization of the variables used to accumulate the probabilities and times for a loop or path. At statement 100 a check is made to see whether all nodes of the loop or path have been considered. If all nodes have not been considered, then the present node number is saved by statement 101 and a check is made on the next node number. If that node value is zero, then the node number is saved by statement 103. At this time, I6 is the start node for the branch and I7 is the end node. I8 tells at what value of NL(·) to start the search and I5 tells how many times the start node, I6, appears in the input network. A check is then made on I5 to see if it is equal to zero. It can only become equal to zero if at statement 102

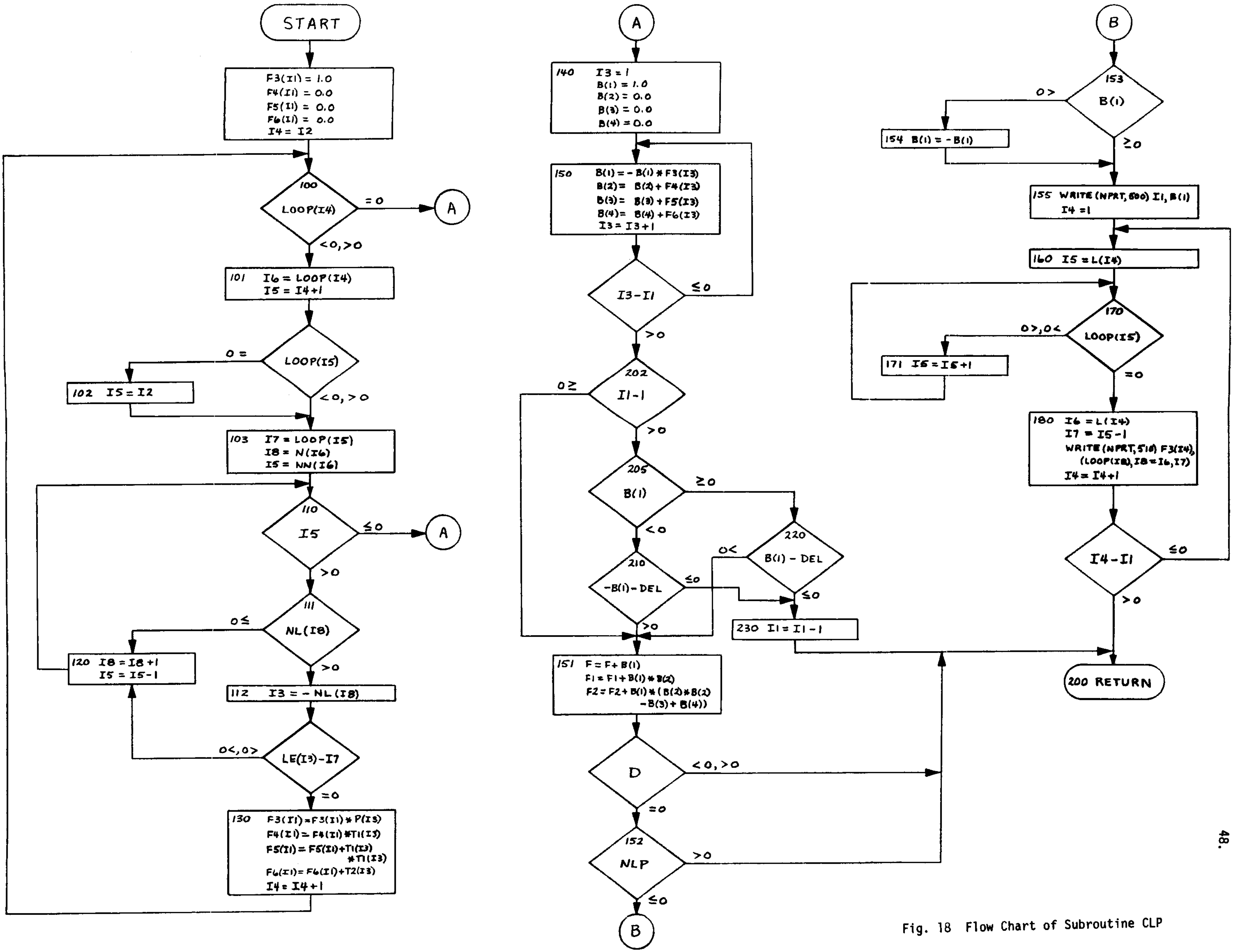



Fig. 18 Flow Chart of Subroutine CLP

the start node was the start node for a path. In that situation, there will be no branch looping back on the start node and I5 will eventually be reduced to zero by the indexing at statement 120. If I5 becomes equal to zero, then transfer is made to point "A" of the flow chart where the path calculations are made. If a loop is being considered instead of a path, then the proper branch between nodes I6 and I7 will be located before I5 becomes equal to zero and the program will continue to statement 130 to make the calculations for a branch. If, in the case of either a loop or a path, the program had not reached a node value of zero before reaching statement 110, then it is in the midst of a loop or path and will discover the proper branch before I5 becomes equal to zero. Again, the program will continue to statement 130 to make the branch calculations. After making the branch calculations, the program returns to statement 100 to check the next path node. If there is a node left to examine, then the program goes to statement 101 and repeats the process that was described above. If the node value is zero at statement 100, then transfer is made to point "A" of the flow chart where the loop calculations are made. Thus if the transfer to point "A" is made from statement 100, the calculations will be for loop values, and if the transfer is made from statement 110, the calculations will be path values.

The appropriate variables are initialized at statement 140 and the calculations are made at statement 150. The check of I3 versus I1 is pertinent primarily to higher order loop calculations and is a check to see that all compound loop products have been calculated. At statement 202, the program checks to see whether a higher order loop is being considered. If not, the final loop or path calculations are completed at statement 151. If the loop is a higher order loop, however, its probability is checked against the deletion probability which is a user input to the program to see if the loop should be considered or not. If it is not to be considered, program control is returned to the calling subroutine. If the higher order loop is to be considered, the final loop calculations are completed at statement 151. A check is made on D to see whether to print the loop values. If D = 0, the loop values are to be printed while if D ≠ 0, control is returned to the calling program. If loops are to be printed, a check is made on NLP at statement 152 to see whether the user wants the loops to be printed. If he does, then the section of the program from point "B" on the flow chart down to the return statement accomplishes the loop printout. The loop printout for the network of Fig. 5 is shown in Fig. 19.

```
LOOP OF ORDER    1      W(O)= 0.119999
          W(O)= 0.1199 , NODES    3    5

LOOP OF ORDER    2      W(O)= 0.025199
          W(O)= 0.1199 , NODES    3    5
          W(O)= 0.2099 , NODES    4    6

LOOP OF ORDER    1      W(O)= 0.007199
          W(O)= 0.0071 , NODES    3    6    4    5

LOOP OF ORDER    1      W(O)= 0.209999
          W(O)= 0.2099 , NODES    4    6
```

Fig. 19 Loop Printout for Network Given in Fig. 5

The FORTRAN statements comprising Subroutine CLP are shown in Fig. 20.

```
      SUBROUTINE CLP
C
C*****COMPOUND LOOP PRODUCTS
C
      COMMON I1,I2,I3,I4,I5,I6,I7,I8,I9,
     1       NNO,NLO,LLO,LPO,NLP,NPP,NCRD,NPRT,JCOR,
     2F,F1,F2,F3(50),F4(50),F5(50),F6(50),P(100),
     3       D,D1,D2,N1,N2,GP,GP1,GP2,T,VT,
     4T1(100),T2(100),LS(100),LE(100),N(100),NN(100),NL(200),L(100)
      COMMON LOOP(1000),NS(50),NE(50),G(50),G1(50),G2(50),K(3),B(8),
     1J(9),NAME(6),NJOB(2),MON,NDY,NYR,DEL
C
C*****SUB. CLP CALCULATES THE PROBABILITIES AND TIMES ASSOCIATED
C*****WITH ALL BRANCHES IN THE LOOP
C
      F3(I1)=1.0
      F4(I1)=0.0
      F5(I1)=0.0
      F6(I1)=0.0
      I4=I2
  100 IF(LOOP(I4))101,140,101
  101 I6=LOOP(I4)
      I5=I4+1
      IF(LOOP(I5))103,102,103
  102 I5=I2
  103 I7=LOOP(I5)
      I8=N(I6)
      I5=NN(I6)
  110 IF(I5)140,140,111
  111 IF(NL(I8))112,120,120
  112 I3=-NL(I8)
      IF(LE(I3)-I7)120,130,120
  120 I8=I8+1
      I5=I5-1
      GO TO 110
  130 F3(I1)=F3(I1)*P(I3)
      F4(I1)=F4(I1)+T1(I3)
      F5(I1)=F5(I1)+T1(I3)*T1(I3)
      F6(I1)=F6(I1)+T2(I3)
      I4=I4+1
      GO TO 100
  140 I3=1
      B(1)=1.0
      B(2)=0.0
      B(3)=0.0
      B(4)=0.0
  150 B(1)=-B(1)*F3(I3)
      B(2)=B(2)+F4(I3)
      B(3)=B(3)+F5(I3)
      B(4)=B(4)+F6(I3)
      I3=I3+1
      IF(I3-I1)150,150,202
  202 IF(I1-1)151,151,205
  205 IF(B(1))210,220,220
C
C*****IF DEL IS GT ZERO, LOOPS WITH PROB. LT DEL ARE DELETED
C
  210 IF(-B(1)-DEL)230,230,151
  220 IF(B(1)-DEL)230,230,151
  230 I1=I1-1
      GO TO 200
  151 F=F+B(1)
      F1=F1+B(1)*B(2)
      F2=F2+B(1)*(B(2)*B(2)-B(3)+B(4))
  500 FORMAT(1H0,1X,13HLOOP OF ORDER,I4,5X,5HW(0)=,F9.6)
  510 FORMAT(9X,5HW(0)=,F7.4,8H , NODES,15I5)
C
C*****IF D = 0, LOOPS ARE BEING EXAMINED AND PRINTOUT OCCURS
C*****IF D IS NOT ZERO, PATHS ARE BEING EXAMINED AND PRINTOUT IS OMITTED
C
      IF(D)200,152,200
C
C*****IF NLP IS GT ZERO, LOOP PRINTOUT IS SUPPRESSED
C
  152 IF(NLP)153,153,200
  153 IF(B(1))154,155,155
  154 B(1)=-B(1)
  155 WRITE(NPRT,500) I1,B(1)
      I4=1
  160 I5=L(I4)
  170 IF(LOOP(I5))171,180,171
  171 I5=I5+1
      GO TO 170
  180 I6=L(I4)
      I7=I5-1
      WRITE(NPRT,510) F3(I4),(LOOP(I8),I8=I6,I7)
      I4=I4+1
      IF(I4-I1)160,160,200
  200 RETURN
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR CLP
 COMMON    3184  VARIABLES      6  PROGRAM     544

END OF COMPILATION
```

Fig. 20 FORTRAN Listing of Subroutine CLP

Subroutine PV

Subroutine PV is called by the main program to calculate the path values for all paths through the network. It first calculates the values associated with the branches of a path and then calculates the values for all loops that are disjoint from the path. The values are then combined through the use of the topology equation to compute the equivalent values for the path. These values are then printed out. The values are also accumulated by this subroutine to aid in the calculation of the equivalent branches of the network which is done by subroutine PRP.

The flow chart for subroutine PV is shown in Fig. 21. The initialization shown in the first box in the chart tells where to locate the first node, L(1), for the first path and the number of equivalent branches in the network, I9. The checks on NPP and NLP are merely to determine whether it is necessary to slew a page and whether to print the path values. At statement 95, the heading for the path printout is printed unless the user option not to print paths is exercised. The portion of the flow chart from statement 100 to the check on NPP following statement 131 represents the actual calculation of path values. The remainder of the subroutine is used to accumulate the values needed by subroutine PRP for calculation of the values for the equivalent branches of the network. Path values are calculated after statement 100. CLP is called and calculates the values associated with the branches in the path. By repeatedly calling CL, the value of disjoint loops are included in the calculation where nedessary.

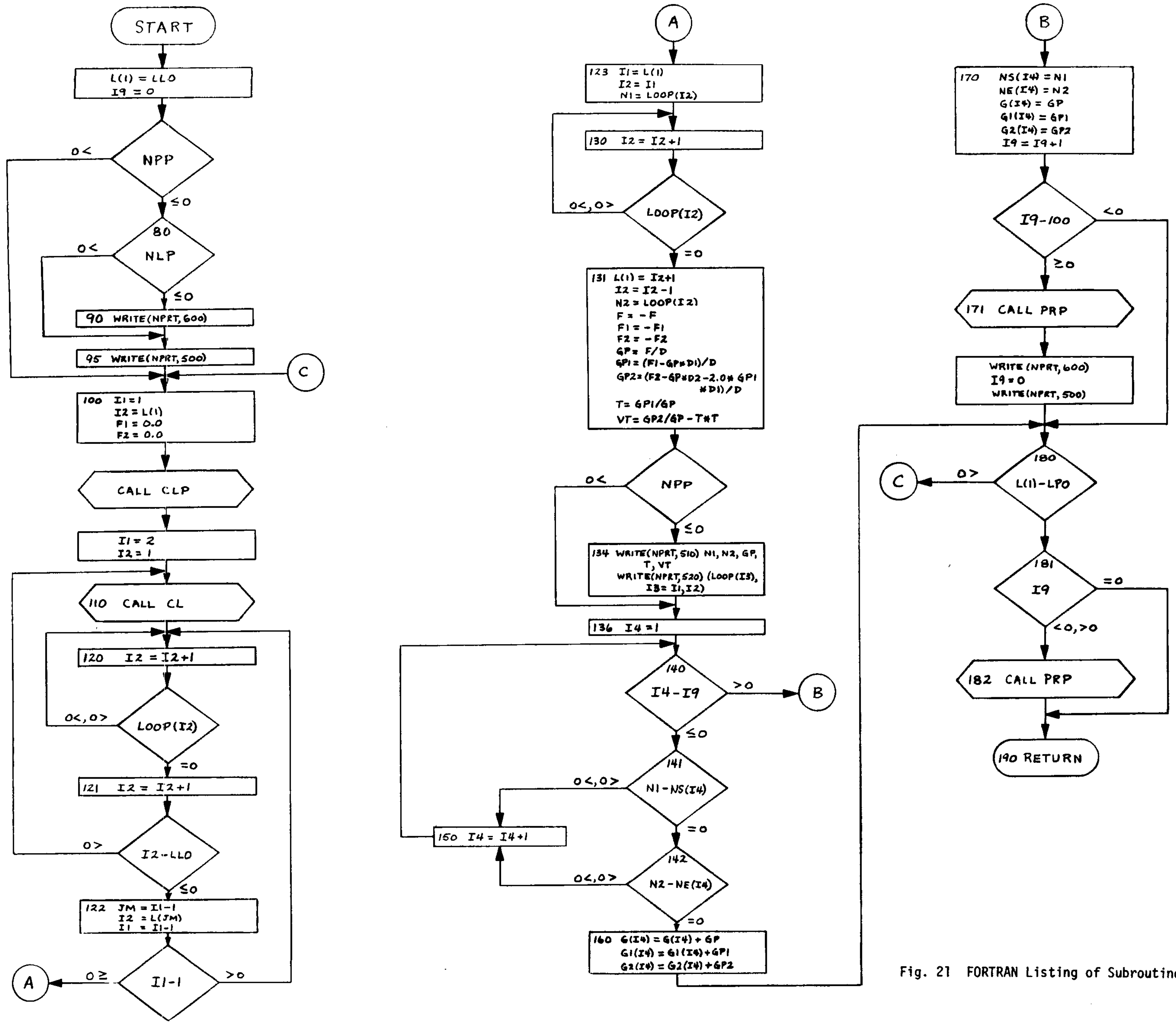



Fig. 21 FORTRAN Listing of Subroutine PV

One should recall that subroutine CL calls subroutine CLP to make the actual calculation of values associated with the loops. Following statement 131, the loop and path values are combined to compute the equivalent values for the path. These values are then printed out by statement 134 unless the option to delete path printout has been exercised.

The portion of the subroutine from statement 136 on is devoted to accumulating values for calculation of the equivalent branch values. The transfer statements from statement 140 to 142 check to see whether the source and sink node for the path just examined are the same as those for the immediately preceding path. Paths are grouped so that all paths with the same source and sink nodes are stored consecutively. If they are both the same, then statement 160 continues to accumulate the values for the equivalent branch. If either node is found to be different, then a new branch is started at statement 170 and the number of equivalent branches, I9, is incremented by one. If fewer than 100 equivalent branches have been located, then the program continues to statement 180 where a check is made to see whether all paths have been located. If not, return is made to statement 100 and the process described above is repeated. If all paths have been found, then subroutine PRP is called to calculate values for the equivalent branches of the network. Upon return from PRP, program control is returned to the main program.

The printout of path values adjusted for loop considerations for the network of Fig. 5 is shown in Fig. 22. The FORTRAN statements comprising subroutine PV are shown in Fig. 23.

| NS | NE | PROB | M T | V T | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7 | 0.551162 | 3.4926 | 19.7059 | 1 | 3 | 5 | 7 | | |
| 1 | 7 | 0.041860 | 18.6191 | 41.2408 | 1 | 3 | 6 | 4 | 5 | 7 |
| 1 | 8 | 0.406976 | 7.6191 | 41.2409 | 1 | 3 | 6 | 8 | | |
| 2 | 8 | 0.024418 | 19.6191 | 41.2409 | 2 | 4 | 5 | 3 | 6 | 8 |
| 2 | 7 | 0.348837 | 8.6191 | 41.2409 | 2 | 4 | 5 | 7 | | |
| 2 | 8 | 0.626744 | 4.3919 | 28.6892 | 2 | 4 | 6 | 8 | | |

Legend:

| Symbol | Definition |
|---|---|
| NS | Source node of path |
| NE | Source node of path |
| PROB | Probability of going from NS to NE |
| MT | Expected time to go from NS to NE |
| VT | Variance of the time to go from NS to NE |
| i j k l | Path with NS = i, NE = l and intermediate nodes j, k |

Fig. 22 Printout of the Paths for Network of Fig. 5

```
      SUBROUTINE PV
C
C*****PATH VALUES
C
      COMMON I1,I2,I3,I4,I5,I6,I7,I8,I9,
     1       NNO,NLO,LLO,LPO,NLP,NPP,NCRD,NPRT,JCOR,
     2F,F1,F2,F3(50),F4(50),F5(50),F6(50),P(100),
     3       D,D1,D2,N1,N2,GP,GP1,GP2,T,VT,
     4T1(100),T2(100),LS(100),LE(100),N(100),NN(100),NL(200),L(100)
      COMMON LOOP(1000),NS(50),NE(50),G(50),G1(50),G2(50),K(3),B(8),
     1J(9),NAME(6),NJOB(2),MON,NDY,NYR,DEL
  500 FORMAT (4X,2HNS,4X,2HNE,3X,4HPROB,7X,3HM T,7X,3HV T)
  510 FORMAT (1X,I5,1X,I5,1X,F9.6,1X,F9.4,1X,F9.4)
  520 FORMAT (44X,15I5)
  600 FORMAT(1H1)
C
C*****INITIALIZE PATH SEARCH
C
      L(1)=LLO
      I9=0
      IF(NPP)80,80,100
   80 IF(NLP)90,90,95
   90 WRITE(NPRT,600)
   95 WRITE(NPRT,500)
  100 I1=1
C
C*****L(1) IS THE FIRST NODE IN A PATH
C
      I2=L(1)
      F=0.0
      F1=0.0
      F2=0.0
C
C*****CLP IS CALLED TO CALCULATE VALUES FOR THE BRANCHES IN THE PATH
C
      CALL CLP
      I1=2
      I2=1
C
C*****CL IS CALLED TO LOCATE DISJOINT LOOPS***IT CALLS CLP TO
C*****CALCULATE VALUES IF DISJOINT LOOPS ARE FOUND
C
  110 CALL CL
  120 I2=I2+1
      IF(LOOP(I2))120,121,120
  121 I2=I2+1
      IF(I2-LLO)110,122,122
  122 JM=I1-1
      I2=L(JM)
      I1=I1-1
      IF(I1-1)123,123,120
  123 I1=L(1)
      I2=I1
      N1=LOOP(I2)
  130 I2=I2+1
      IF(LOOP(I2))130,131,130
C
C*****PATH VALUES ARE CALCULATED AND PRINTED OUT
C
  131 L(1)=I2+1
      I2=I2-1
      N2=LOOP(I2)
      F=-F
      F1=-F1
      F2=-F2
      GP=F/D
      GP1=(F1-GP*D1)/D
      GP2=(F2-GP*D2-2.0*GP1*D1)/D
      T=GP1/GP
      VT = GP2/GP-T*T
```

Fig. 23 FORTRAN Listing of Subroutine PV

```
C
C*****IF NPP IS GT ZERO, PATH PRINTOUT IS SUPPRESSED
C
      IF(NPP)134,134,136
  134 WRITE(NPRT,510) N1,N2,GP,T,VT
      WRITE(NPRT,520) (LOOP(I3),I3=I1,I2)
  136 I4=1
  140 IF(I4-I9)141,141,170
C
C*****DETERMINE START AND END NODE FOR THE PATH
C
  141 IF(N1-NS(I4))150,142,150
  142 IF(N2-NE(I4))150,160,150
  150 I4=I4+1
      GO TO 140
C
C*****START AND END NODE ARE THE SAME AS A PREVIOUS PATH***CONTINUE
C*****TO ACCUMULATE VALUES FOR PATHS WITH THESE SAME NODES
C
  160 G(I4)=G(I4)+GP
      G1(I4)=G1(I4)+GP1
      G2(I4)=G2(I4)+GP2
      GO TO 180
C
C*****START OR END NODE IS NOT THE SAME AS THE PREVIOUS PATH***BEGIN
C*****TO ACCUMULATE THE VALUES FOR THE NEW SET OF NODES
C
  170 NS(I4)=N1
      NE(I4)=N2
      G(I4)=GP
      G1(I4)=GP1
      G2(I4)=GP2
      I9=I9+1
      IF(I9-100)180,171,171
  171 CALL PRP
      WRITE(NPRT,600)
      I9=0
      WRITE(NPRT,500)
  180 IF(L(1)-LPO)100,181,181
  181 IF(I9)182,190,182
C
C*****ALL PATHS HAVE BEEN EXAMINED
C
  182 CALL PRP
  190 RETURN
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR PV
 COMMON    3184  VARIABLES      6  PROGRAM     476

END OF COMPILATION
```

Fig. 23 FORTRAN Listing of Subroutine PV (continued)

Subroutine PRP

Subroutine PRP is called by subroutine PV to calculate and print the values for the equivalent branches of the network. It is within PRP that loop values may be deleted and normalization of the final values may take place.

The flow chart for subroutine PRP is shown in Fig. 24. Initialization occurs in the first box of the chart. If either DEL or JCOR are zero or less, the printout is not adjusted. If DEL is greater than zero, then there is a possibility that the sum of the probabilities for the equivalent branches emanating from a particular source node may not equal one. In this situation, the user can specify by the use of JCOR whether or not he wishes these probabilities to be adjusted to sum to one. If the option is exercised, then the adjustment for an equivalent branch probability is simply 1/GT where GT is the sum of the probabilities for all equivalent branches emanating from a given source node. The branch times are adjusted by the same amount.

The situation will first be considered where the values are to be left unchanged. The checks on NPP and NLP are for page control and heading printout control. The problem heading is printed by statement 100 if NLP $\leq$ 0. A DO loop is used to print 50 equivalent branches to a page of output. For the no adjustment case, I1 is always less than I7 and transfer is made to statement 60 where the equivalent branch values are computed and printed. The above process is repeated until all branches have been printed. Program control then returns to the calling subroutine which was subroutine PV.

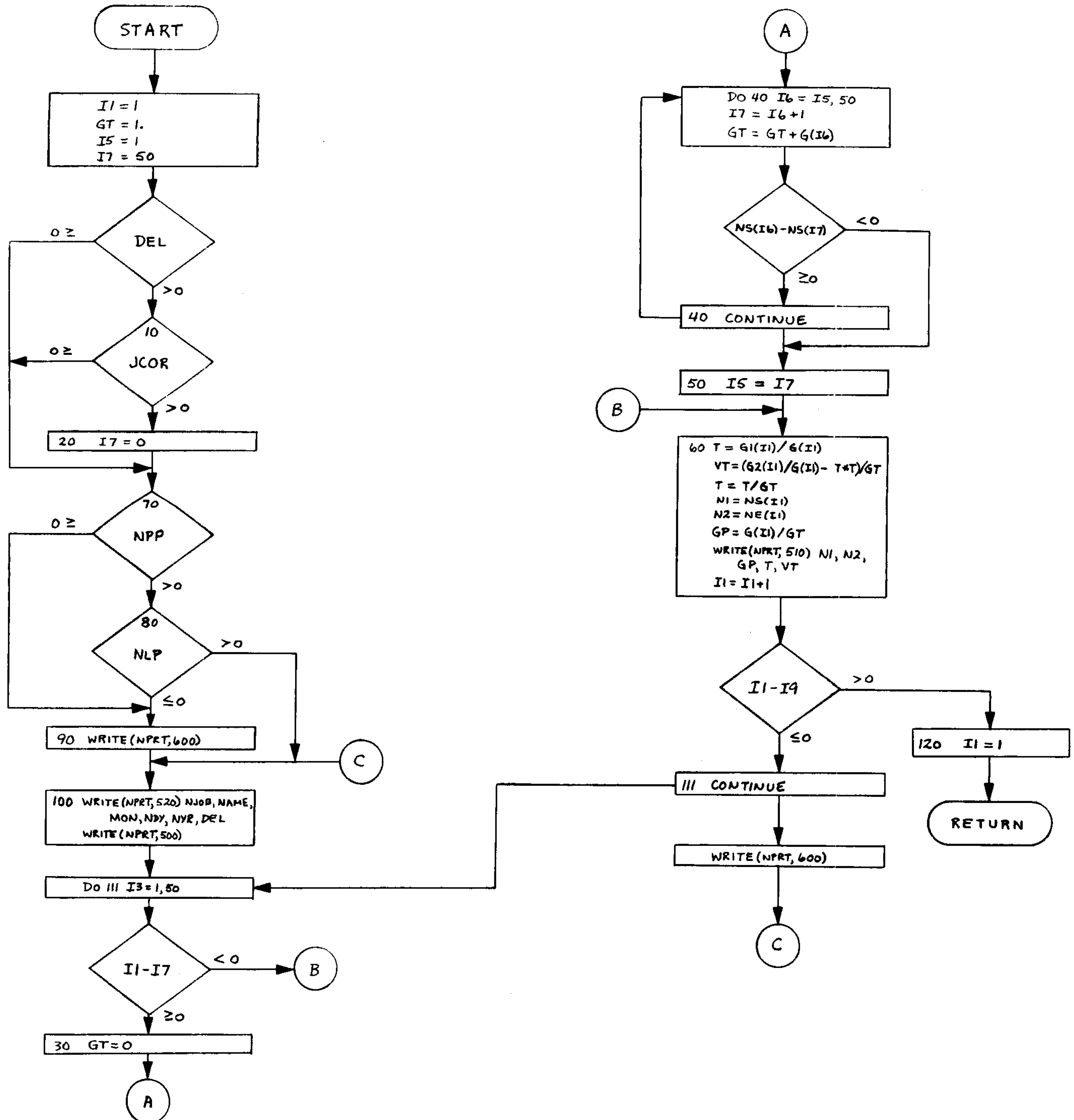



Fig. 24 Flow Chart of Subroutine PRP

For the normalization case, I7 is set equal to zero at statement 20. The process continues as stated above down to check on I1 versus I7. I1 represents the number of the branch to be computed and printed and I7 is one greater than the number of the last branch emanating from a given source node. When $I1 < I7$, transfer is made to statement 60 to calculate and print the branch. When $I1 \geq I7$, a new source node is to be considered and the probabilities for the branches are summed by the portion of the subroutine from statement 30 to statement 40. At statement 50, the starting branch number for the next source node is saved. The program then goes on to statement 60 to compute and print the branch values. As before, the process is repeated until all equivalent branches have been printed. Program control then returns to subroutine PV.

The equivalent branches for three modules of the network of Fig. 5 are shown in Fig. 25 for the cases: 1) no loop deletions; 2) loops with probability less than .0001 deleted; and 3) loops with probability less than .0001 deleted and final outputs normalized. The FORTRAN statements comprising subroutine PRP are shown in Fig. 26.

GERT PROBLEM NO. 1A BY PHIL ISHMAEL DATE 6/ 11/ 1968

LOOP DELETION VALUE, DEL = 0.00000000

EQUIVALENT BRANCHES OF THE NETWORK

| ENTRY | EXIT | PROBABILITY | MEAN TIME | VARIANCE |
|---|---|---|---|---|
| 1 | 27 | 0.469389E 00 | 0.177158E 02 | 0.127568E 03 |
| 1 | 28 | 0.530634E 00 | 0.178962E 02 | 0.120472E 03 |
| 2 | 28 | 0.545194E 00 | 0.179425E 02 | 0.125641E 03 |
| 2 | 27 | 0.454829E 00 | 0.188962E 02 | 0.120472E 03 |

a) No Loop Deletions

GERT PROBLEM NO. 1D BY PHIL ISHMAEL DATE 6/ 11/ 1968

LOOP DELETION VALUE, DEL = 0.00010000

EQUIVALENT BRANCHES OF THE NETWORK

| ENTRY | EXIT | PROBABILITY | MEAN TIME | VARIANCE |
|---|---|---|---|---|
| 1 | 27 | 0.470503E 00 | 0.178136E 02 | 0.132018E 03 |
| 1 | 28 | 0.531022E 00 | 0.179398E 02 | 0.123131E 03 |
| 2 | 28 | 0.545211E 00 | 0.179658E 02 | 0.127725E 03 |
| 2 | 27 | 0.455162E 00 | 0.189398E 02 | 0.123131E 03 |

b) Loops Having a Probability Less Than 0.0001 Deleted

GERT PROBLEM NO. 1F BY PHIL ISHMAEL DATE 6/ 11/ 1968

LOOP DELETION VALUE, DEL = 0.00010000

EQUIVALENT BRANCHES OF THE NETWORK

| ENTRY | EXIT | PROBABILITY | MEAN TIME | VARIANCE |
|---|---|---|---|---|
| 1 | 27 | 0.469786E 00 | 0.177865E 02 | 0.131816E 03 |
| 1 | 28 | 0.530213E 00 | 0.179125E 02 | 0.122944E 03 |
| 2 | 28 | 0.545007E 00 | 0.179591E 02 | 0.127677E 03 |
| 2 | 27 | 0.454992E 00 | 0.189328E 02 | 0.123085E 03 |

c) Loops Deleted, Values Normalized

Fig. 25. Final Outputs From GERT Program

```
      SUBROUTINE PRP
C
C*****PRINT PATHS
C
      COMMON I1,I2,I3,I4,I5,I6,I7,I8,I9,
     1       NNO,NLO,LLO,LPO,NLP,NPP,NCRD,NPRT,JCOR,
     2F,F1,F2,F3(50),F4(50),F5(50),F6(50),P(100),
     3       D,D1,D2,N1,N2,GP,GP1,GP2,T,VT,
     4T1(100),T2(100),LS(100),LE(100),N(100),NN(100),NL(200),L(100)
      COMMON LOOP(1000),NS(50),NE(50),G(50),G1(50),G2(50),K(3),B(8),
     1J(9),NAME(6),NJOB(2),MON,NDY,NYR,DEL
  500 FORMAT (1X,5HENTRY,2X,4HEXIT,3X,11HPROBABILITY,5X,9HMEAN TIME,6X,
     1       8HVARIANCE)
  510 FORMAT (1X,I5,1X,I5,3E15.6)
  520 FORMAT(18H GERT PROBLEM NO. ,2A2,6H   BY   ,6A2,6H   DATE,I3,
     11H/,I3,1H/,I5//11X,26HLOOP DELETION VALUE, DEL =,F11.8///
     212X,34HEQUIVALENT BRANCHES OF THE NETWORK/)
  600 FORMAT (1H1)
      I1=1
      GT=1.
      I5=1
      I7=50
      IF(DEL)70,70,10
   10 IF(JCOR)70,70,20
C
C*****IF DEL AND JCOR ARE BOTH GT ZERO, FINAL EQUIVALENT BRANCHES WILL
C*****BE ADJUSTED SO THAT PROBS. ASSOCIATED WITH START NODES SUM TO ONE
C
   20 I7=0
   70 IF(NPP)90,90,80
   80 IF(NLP)90,90,100
   90 WRITE(NPRT,600)
  100 WRITE(NPRT,520) NJOB,NAME,MON,NDY,NYR,DEL
      WRITE(NPRT,500)
      DO 111 I3=1,50
      IF(I1-I7)60,30,30
   30 GT=0
      DO 40 I6=I5,50
      I7=I6+1
      GT=GT+G(I6)
      IF(NS(I6)-NS(I7))50,40,40
   40 CONTINUE
   50 I5=I7
C
C*****CALCULATE AND PRINT ALL VALUES FOR THE EQUIV. NETWORK BRANCHES
C
   60 T=G1(I1)/G(I1)
      VT=(G2(I1)/G(I1)-T*T)/GT
      T=T/GT
      N1=NS(I1)
      N2=NE(I1)
      GP=G(I1)/GT
      WRITE(NPRT,510) N1,N2,GP,T,VT
      I1=I1+1
      IF(I1-I9)111,111,120
  111 CONTINUE
      WRITE(NPRT,600)
      GO TO 100
  120 I1=1
      RETURN
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR PRP
 COMMON    3184  VARIABLES       8   PROGRAM     354

END OF COMPILATION
```

Fig. 26 FORTRAN Listing of Subroutine PPP

## Relationships Between the Dimensioned Variables

The relationships between the dimensioned variables will be discussed with regard to two conditions: across-the-board program sizing and tailored dimensioning. For across-the-board sizing, the primary consideration is to establish a set of variable dimensions that accommodate the largest possible problem for a given computer. For example, it is not considered feasible to attempt to accommodate an input network in excess of 100 branches for the GE 225 and the IBM 1130 in the Arizona State University computer center. However, it might be possible to tailor the program dimensions to accommodate specific networks with more than 100 branches. Such tailoring would depend on the configuration of the input network.

### Across-the-Board Program Sizing

To be able to read into the program and manipulate a 100-branch network, the following variables must be dimensioned at 100:

| | | | |
|---|---|---|---|
| LS(100) | P(100) | T2(100) | NN(100) |
| LE(100) | T1(100) | N(100) | L(100) |

These dimensions are required since:

1. LS and LE contain the start and end node for each branch;
2. P, T1, and T2 are the probability, first moment of time to traverse the branch, and second moment of time to traverse the branch, respectively.
3. N and NN corespond to each node number of the input network (it is necessary to use node numbers between 1 and 100, inclusive, for networks approaching 100 branches in size. As is indicated in the user's manual, however, there is no restriction on the order in which the node numbers appear in the input network.)

4. L is the greater of the largest number of nodes contained in any first order loop plus one or the largest number of nodes appearing in a path. L is dimensioned rather generously since the largest subscript for L ever actually used by the program could only approach 100 if all branches of the input network were in series.

The only other dimensioned variable that is directly related to a 100-branch input network is NL(·) which is dimensioned at 200. The dimension for NL must always be twice the size of the largest allowable number of input branches since there is an NL value for each end of the branch.

The variables F3(50),F4(50), F5(50), and F6(50) are actually related to the highest order loop that can be expected rather than the number of branches in the input network. They are dimensioned rather generously insofar as a 100-branch input network is concerned.

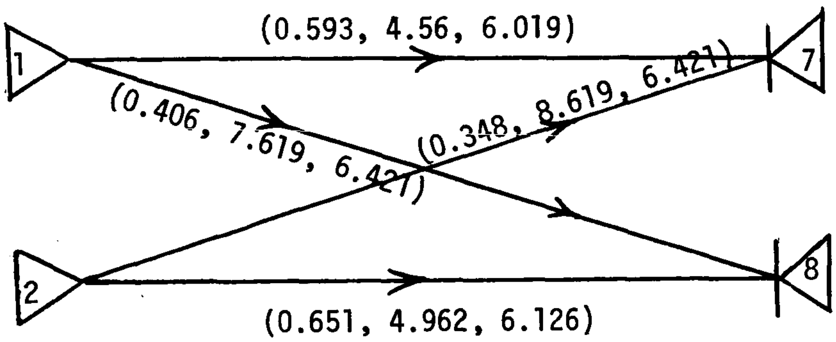
The variables NS(50), NE(50), G(50), G1(50), and G2(50) are related to the number of possible equivalent branches that can be accommodated by the program. If $S_S$ represents the number of source nodes in the input network and $S_E$ represents the number of sink nodes and if any sink node can be reached from any source node, then the maximum allowable number of such nodes must meet the following restriction: $S_S * S_E \leq 50$. For the input network of Fig. 5 and Table 1, there are only two source nodes and two sink nodes; therefore, the largest subscript for NS, NE, G, G1, and G2 that was actually required for the example input network was four.

The dimensioned variable whose size is the most critical for an input network of any appreciable size is LOOP(·) which is dimensioned as 1000. This variable is used to record each node appearing in all first order loops and each node in each path through the network. LOOP(·) values of zero separate each loop and each path. If an input network has many possible paths and, in particular, if each path contains many branches, then the dimension on the variable LOOP(·) can rapidly grow large and may get larger than the specified dimension for the program. Such a situation is described in the following discussion of tailored dimensioning of the program.

## Tailored Dimensioning

The term "tailored dimensioning" is used here to describe the process of altering the GERT EXCLUSIVE-OR program to fit a specific input network. The alteration is accomplished, for the most part, by varying the dimensions of the variables that are in COMMON. Usually the reason that a problem will not fit into the standard GERT program is that the dimension for the variable LOOP(·) has become larger than is specified in the program. Thus, the primary goal of tailoring the program is to enlarge the dimension for LOOP(·) at the expense of other variables whose dimensions do not need to be as large as they are in the standard GERT EXCLUSIVE-OR program.

To illustrate how such tailoring might be accomplished, a portion of Example 1 from the user's manual (2) will be used. In that example, a four-module problem was discussed where each module was like the network shown in Fig. 5 of this report. The input network for the four-module problem contains only forty branches, but the dimension on the variable LOOP (·) grows to 2746, which is 1746 words larger than the standard GERT IBM 1130 program can accommodate. Since there are only 40 branches, then the variables LS, LE, P, T1, T2, N, and NN could be dimensioned at 40 each which releases (7)(60) = 420 decimal words of storage. The variable L could safely be reduced to a dimension of 25 which releases 75 more words of storage. Since there are only two source nodes and two sink nodes, the variables NS, NE, G, G1, and G2 could each be dimensioned at 4 which releases (5)(46) = 230 additional words of storage. Since the highest order loop that is possible for the network is an eighth order loop, the variables F3, F4, F5 and F6 could safely be dimensioned at 15 which releases another (4)(35) = 140 decimal words of storage. At the standard GERT program dimensions, there are an additional 258 decimal words of storage available on the IBM 1130 which could also be used. Thus, the number of additional words of storage that can be made available by tailoring the standard program to fit the four-module network problem is 1123. The largest dimension for LOOP(·) could be specified as 2123 which is still too small to handle the network. At this point, two alternatives are available: 1. use a larger machine or pack the values of LOOP(·); 2. analyze the network in segments. Alternative 1 is not always a possibility. For alternative 2, one way of breaking the four-module problem into segments is to make a pass on the computer to obtain the equivalent branches of the network for one module. The equivalent branches for the network of Fig. 5 are shown below:



In the above network the numbers in parentheses associated with each branch indicate the probability, the mean time, and the standard deviation of time to traverse the branch. When segmenting a network, the values of each segment can be inputted using a normal distribution of time (if only two moments are used). By using two such equivalent network modules and two modules like Fig. 5, it was possible to run the four-module problem on the standard IBM 1130 GERT program.

As was stated previously, the $n^{th}$ moment of the equivalent network only depends on first n moments of the branches. By reducing a complex network in segments (if this is possible) and describing the reduced branches in terms of the first n moments, a large network can be analyzed. The four-module network was run without segmentation on the CDC 3400. As expected, the results from the IBM 1130 run and the CDC 3400 run were identical.

BIBLIOGRAPHY ON GERT AND RELATED TOPICS

1. Brock, P. and S. M. Drezner, "A Problem Emanating from a Study of Feedback Loops in Stochastic Networks," The RAND Corporation, P 3276, February, 1966.

2. Deutsch, D., "The Theory of Blocking," Masters Report, Arizona State University, May, 1966.

3. Drezner, S. M. and A. A. B. Pritsker, Network Analysis of Countdown, The RAND Corporation, RM-4976-NASA, March, 1966.

4. Eisner, Howard, "A Generalized Network Approach to the Planning and Scheduling of a Research Program," Journal of Operations Research Society, Vol. 10, No. 1, 1962.

5. Elmaghraby, Salah E., "An Algebra for the Analysis of Generalized Activity Networks," Management Science, Vol. 10, No. 3, April, 1964, pp. 494-514.

6. _______, "On Generalized Activity Networks," Journal of Industrial Engineering, Vol. XVII, No. 11, November, 1966, pp. 621-631.

7. Graham, P., "Profit Probability Analysis of Research and Development Expenditures," The Journal of Industrial Engineering, Vol. XVI, No. 3, May-June, 1965, pp. 186-191.

8. Happ, W. W., "Flowgraph Techniques for Closed Systems," IEEE Transactions, AES, Vol. 2, May, 1966, pp. 252-264.

9. Hill, T. W., "System Optimization: A Sensitivity Approach Using GERT," unpublished Masters Research Report, Arizona State University, Tempe, Arizona, 1966 (Also in Bulletin #3, Industrial Engineering Research at Arizona State University, 1966).

10. Ishmael, P. C. and A. A. B. Pritsker, User Manual for GERT EXCLUSIVE-OR Program, NASA/ERC, NGR 03-001-034, July, 1968.

11. _______, __________, Definitions and Procedures Employed in the GERT EXCLUSIVE-OR Program, NASA/ERC, NGR 03-001-034, July, 1968.

12. Ledbetter, L. R., "A Study of Computational Aspects of Network Models for Planning and Control," Masters Thesis, Lehigh University, 1967.

13. Powell, G. E., "Development, Evaluation and Selection of a Dodge Continuous Sampling Plan When the Rectifying Operation is Not Perfect," Masters Thesis, Lehigh University, 1967

14. McDonald, W. W., "Analysis of Stochastic Networks," in GRAPH THEORY, Significant Books in Operations Research, Management Information Services, Detroit, Michigan (Also Masters Thesis, Arizona State University, 1966.)

15. Pritsker, A. A. B., GERT: Graphical Evaluation and Review Technique, The RAND Corporation, RM-4973-NASA, April, 1966.

16. ______, and W. W. Happ, "GERT: Graphical Evaluation and Review Technique, Part I Fundamentals," The Journal of Industrial Engineering, Vol. XVII, No. 5, pp. 267, May, 1966.

17. ______, and G. E. Whitehouse, "GERT: Graphical Evaluation and Review Technique, Part II Probabilistic and Industrial Engineering Applications," Journal of Industrial Engineering, Vol, XVII, No. 6, June, 1966.

18. Smith, R. L., "Stochastic Analysis of Personnel Movement in Formal Organization," Ph.D. Dissertation, Arizona State University, August, 1967.

19. Werberger, G., "Optimality Considerations of GERT," Masters Thesis, Lehigh University, 1967.

20. Whitehouse, Gary E., "Extensions, New Developments, and Applications of GERT," Ph.D. Dissertation, Arizona State University, August, 1965.