SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

158001

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA

(305) - 284-6257

SHARE PROGRAM LIBRARY SUBMITTAL FORM

SHARE PROGRAM LIBRARY AGENCY
Triangle Universities Computation Center
Post Office Box 12076
Research Triangle Park, North Carolina USA 27709

SPLA CONTROL NUMBER: 00236

This form should be completed and submitted with the program package to the SHARE Program Library Agency at the address shown above. Standards and instructions for submitting programs are in the SHARE Reference Manual, Section 6.

(1) Program Number (to be filled by SPLA) . . . . . SDA 3536 360D-15.8.001

(2) Title of Program . . . . . Out of Kilter Network Routine

(3) System Type(s) (Machine) . . . . . IBM 370/145

(4) Search Key(s) . . . . .

(5) Programming Systems/Languages . . . . Fortran IV

(6) Primary Subject Code H3

(7) Minimum System Requirements

(8) New (N) or Revision (R) (if revision, show prior Program Number in Item 1) R

(9) Date of Submittal 6/25/79

(10) Documentation (number of original pages submitted) 37

(11) Author's Name and Address . . . . . R. J. Clasen

(12) Direct Technical Inquiries to Name & Address (if different than Author)
J. K. Peterson
810 S. Flower Street
M.L. 1035
Los Angeles, CA 90017

(13) Submitter's Installation Membership Code . . . . .

(14) Abstract (should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.

SPLA 1001 Revised 11-76

X

SHARE PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide:

a. Purpose
b. Programming Language used
c. Version and modification level or release number
d. Field of application
e. Type of routine (main program, subroutine, etc.)
f. Specific description of machine requirements

AN INDEPENDENT ROUTINE TO SOLVE CAPACITATED NETWORK FLOW PROBLEMS USING A METHOD IN WHICH A MEASURE OF OPTIMALITY IS NOT WORSENED ON ANY ITERATION. FLOWS HAVE UPPER AND LOWER BOUNDS WHICH MAY BE POSITIVE OR NEGATIVE. NO INITIAL FEASIBLE SOLUTION IS NEEDED. HAS PROVISION FOR SOLVING PROBLEMS WHICH VARY SLIGHTLY FROM PREVIOUSLY SOLVED PROBLEMS IN MINIMAL MACHINE TIME. SOURCE LANGUAGE IS FORTRAN IV.

(Please attach additional pages if necessary) . . . . . Total pages attached 2

An "Acknowledgement of Assistance" statement must be attached to this Submittal Form.

Permission to Publish

"I hereby give the SHARE Program Library Agency permission to reprint, reproduce, and distribute this program"

(15) Signature of Submitter and Date _____ 6/28/79

(15) Signature of Installation Addressee _____

SPLA 1001 (R) Revised 11-76

TAPE KEY

NO LABEL

This volume contains 2 files and 3 tape marks arranged as follows:

File 1: Program source deck
EBCDIC
764 card images blocked 77 per block
T/M

File 2: DATA INPUT
EBCIDIC
1705 card images blocked 77 per block
T/M
T/M

The standard IBM program that can read the tape is IEBGENER.

```
//READ       JOB (ROOO,04),ROBERTS,CLASS=G,MSGCLASS=A,
//           TYPRUN=HOLD
//STEP1      EXEC PGM=IEBGENER
//SYSPRINT   DD SYSOUT=A
//SYSUT1     DD DSN=KILTER,UNIT=TAPE,LABEL=(1,NL),
//           VOL=SER=T0225l,DISP=OLD,
//           DCB=(RECFM=FB,LRECL=80,BLICSIZE=6160,DEN=3)
//SYSUT2     DD SYSOUT=A,DCB=(RECFM=FB,LRECL=80,BLIGSIZE=6160)
//SYSIN      DD DUMMY
//
```

The above program will read the first file and print it.

Purpose

The program solves capacitated network problems -- the problem of finding $x_1, x_2, \ldots, x_n$ that minimize

$$\sum_{j=1}^{n} c_j x_j$$

while satisfying $\sum_{j=1}^{n} a_{ij} x_j = b_i \quad i = 1, 2, \ldots, m \qquad (1)$

and $l_j \leq x_j \leq u_j \quad j = 1, 2, \ldots, n \qquad (2)$

where b_i, u_j, l_j and c_j are signed integers of magnitude less than 10^9 with $\sum_{i=1}^{m} b_i = 0$ and each column of the (a_{ij}) matrix is null except for one +1 entry and one -1 entry.

Method

Each constraint (1) corresponds to a unique six-character symbol known as a node. Each variable j corresponds to an ordered pair of nodes (non-unique and non-exhaustive), known as an arc, such that the first (source) node, s, is the index for which $a_{sj} = 1$ and the second (sink) node, t, is the index for which $a_{tj} = -1$. The program uses a slight variant of the method described by D. R. Fulkerson in Journal SIAM, Vol. 9, p. 18, (March 1961).

OUT OF KILTER PROGRAM

BY

R. J. CLASEN

FOR

IBM 370/145

The program solves an n arc problem by inputting an n + 5 card data deck with cards punched as follows (beginning in column 1):

```
BEGIN
Title card for problem
ARCS
   arc data cards (n cards)
END
SOLVE
```

Each of the n arc data cards is punched as follows:

cols. 1 - 6 blank

7 - 12 name of node s

13 - 18 name of node t

19 - 20 blank

21 - 30 c_{st} (Unit cost)

31 - 40 u_{st} (upper capacity)

41 - 50 l_{st} (lower capacity)

51 - 60 x^o_{st} (nominal flow)

The x^o_{st} must satisfy (1) but not (2). The x^o_{st} implicitly determine b_i and also provide an initial starting point for the algorithm. The printed output is in the same form as the input, except that x^o_{st} has been replaced by the solution.

Usage

A source FORTRAN IV deck which will run on the IBM 370/145, has provided. A test problem is part of the deck. The program is punched in EBCDIC characters. The program unambiguously recognizes 6-character node names on the 370. The symbol KI defined in the main program is the FORTRAN number of the input device and KO is the output device.

I. INTRODUCTION

This is an independent routine for solving a general capacitated network problem using the method described by D. R. Fulkerson in the Journal SIAM, Vol. 9, No. 1 (March 1961), pp. 18-27. It is written in FORTRAN IV with Hollerith punching. The deck must be compiled by each installation, adjusting system control cards, dimensions, and input-output unit numbers as described later in the section on compiling the program. The program uses the fast method of computation which was used in the partly machine (7090) language code, now obsolete, SHARE Distribution 1084. Tricks were avoided in this distribution, in order to allow it to be adapted to as many machines as possible. Dimensions in the program were deliberately made smaller than necessary in accordance with the above objective. The program was checked out with several test problems, but it is not used for production, in the included form, at our installation and we would prefer that inquiries be kept to a minimum, and in any event should not be made more than a year after this distribution is issued. We encourage large users of this method to submit their modifications to SHARE (or other organizations) when they are successful in using it with their machines. Permission is

OUT OF KILTER NETWORK ROUTINE

CONTENTS

granted to anyone who submits such a program to copy any part of this write-up, with no acknowledgement necessary.

II. FORMULATION OF PROBLEM

The network consists of: (1) a collection of nodes, which are given names of 6 alphanumeric characters, and numbered by the program internally from 1 through m.

(2) A collection of arcs, each arc being an ordered pair of nodes, say (i,j) with which are associated four quantities c_{ij}, x_{ij}, u_{ij}, ℓ_{ij}. The collection of arcs will not generally include all possible combinations of nodes (non-exhaustive) and may include the same ordered pair of nodes more than once (non-unique). For the arc (i,j), i is called the source node and j is called the sink node.

The capacitated network flow problem is the problem of determining the x_{ij} such that:

1) $\ell_{ij} \leq x_{ij} \leq u_{ij}$ for all (i,j)

2) the net flow into any node (usually zero) remains fixed

3) $\sum_{(i,j)} c_{ij}x_{ij}$ is minimized with respect to solutions that satisfy (1) and (2).

This problem is solved by the program using the following card input:

BEGIN

Title card

ARCS

(data cards)

END

SOLVE

The cards denoted BEGIN, ARCS, END, and SOLVE are punched beginning in column 1. The title card is the heading that is desired on the output. The title card should have column 1 blank, and columns 2-6 must have at least one (non-blank) punch. The data cards consist of one card per arc. The arc (i,j) is punched in the following format:

Format of ARC Data Card

Col.	1 - 6	blank.
	7 - 12	name of the source node i. All different combinations of six characters including blanks are different names.
	13 - 18	name of the sink node j.
	19 - 20	not used (blank).
	21 - 30	c_{ij}, the unit cost of sending from i to j along this arc.

Format of ARC Data Card cont.

Col. 31 - 40 u_{ij}, the upper bound or maximum flow permitted in this arc.

41 - 50 ℓ_{ij}, the lower bound or minimum flow permitted in this arc.

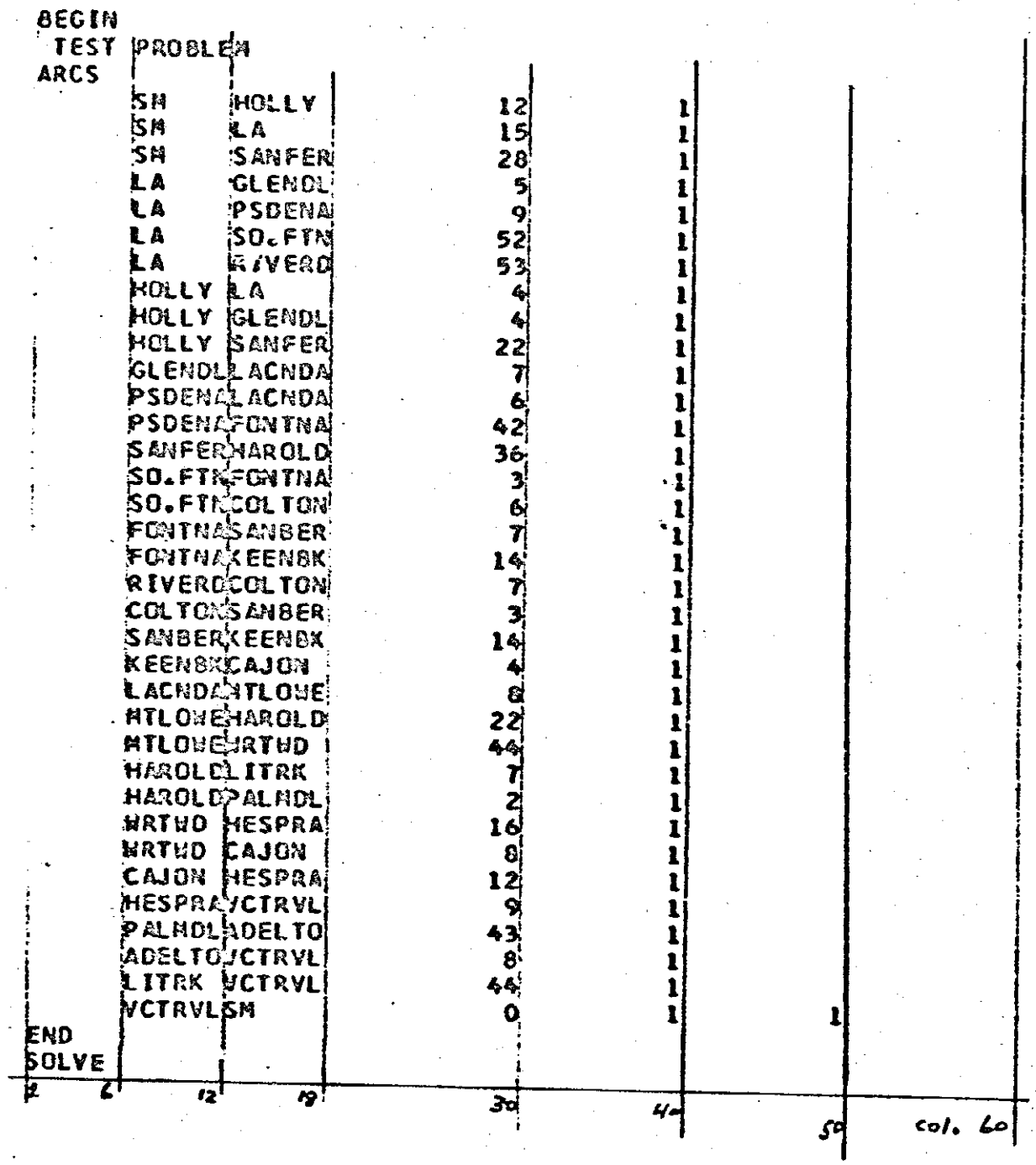
51 - 60 x_{ij}, the initial (nominal) value for the flow in this arc, usually zero. It is used as a guess to the final solution, and to find the net flow into the nodes.

The numbers in columns 21-60 are integers (negative integers are allowed) and must be punched right adjusted in their fields. These integers should not exceed the largest integer representable in FORTRAN. (For a 32-bit machine, the largest acceptable integer is 2,147,483,285.) Furthermore, this data must be ordered so that all cards containing the same name for the source node (in columns 7-12) are adjacent to each other.

With the restrictions noted above, this is all that is needed to solve a network problem. The output will be printed, and the card setup may be followed by another run. As an example, consider the test problem in Figure 1. This is the card setup for a typical problem. The output for this problem is given in Figure 2. The "answer" is in the column headed "FLOW." The column headed "CBAR" is the

Figure 1

TEST PROBLEM



The computer recognizes blank numerical fields as zeros.

Figure 2

SAMPLE PROBLEM OUTPUT

TEST PROBLEM

ARCS		COST	UPPER	LOWER	FLOW	CBAR
SM	HOLLY	12	1	-0	1	-0 K
SM	LA	15	1	-0	-0	-0 K
SM	SANFER	28	1	-0	-0	-0 K
LA	GLENDL	5	1	-0	-0	4 K
LA	PSDENA	9	1	-0	-0	-0 K
LA	SO.FTN	52	1	-0	-0	-0 K
LA	RIVERD	53	1	-0	-0	-0 K
HOLLY	LA	4	1	-0	-0	1 K
HOLLY	GLENDL	4	1	-0	1	-0 K
HOLLY	SANFER	22	1	-0	-0	6 K
GLENDL	LACNDA	7	1	-0	1	-0 K
PSDENA	LACNDA	6	1	-0	-0	7 K
PSDENA	FONTNA	42	1	-0	-0	-0 K
SANFER	HAROLD	36	1	-0	-0	11 K
SO.FTN	FONTNA	3	1	-0	-0	4 K
SO.FTN	COLTON	6	1	-0	-0	-0 K
FONTNA	SANBER	7	1	-0	-0	-0 K
FONTNA	KEENBK	14	1	-0	-0	-0 K
RIVERD	COLTON	7	1	-0	-0	2 K
COLTON	SANBER	3	1	-0	-0	3 K
SANBER	KEENBK	14	1	-0	-0	7 K
KEENBK	CAJON	4	1	-0	-0	1 K
LACNDA	MTLOWE	8	1	-0	1	-0 K
MTLOWE	HAROLD	22	1	-0	-0	-0 K
MTLOWE	WRTWD	44	1	-0	1	-0 K
HAROLD	LITRK	7	1	-0	-0	-0 K
HAROLD	PALMDL	2	1	-0	-0	-0 K
WRTWD	HESPRA	16	1	-0	1	-0 K
WRTWD	CAJON	8	1	-0	-0	-0 K
CAJON	HESPRA	12	1	-0	-0	4 K
HESPRA	VCTRVL	9	1	-0	1	0 K
PALMDL	ADELTO	43	1	-0	-0	-0 K
ADELTO	VCTRVL	8	1	-0	-0	6 K
LITRK	VCTRVL	44	1	-0	-0	4 K
VCTRVL	SM	0	1	1	1	100 K

(The minuses will not print in front of the zeros on some machines)

reduced cost, $c_{ij} + \pi_i - \pi_j$, that is used in the algorithm. The "K" on each line indicates that the problem is "in kilter."

The second basic type of run is the "save and alter" run. This is used when one wishes to change a few arcs of a run without reading in an entire set of arcs. It may be any run but the first. The card setup for this run is as follows.

Title card

SAVE

(alter data cards)

SOLVE

The title card is optional and if present it must precede the "SAVE" card. This run saves the previous run and modifies the arc data of the previous run with the values in the alter data cards.

An alter data card may alter data only for an arc that is already present. There is no way to augment the network with new arcs. The format of the alter data card is as follows:

Format of Alter Data Card

Col.	1 - 5	the punches "ALTER" or five blanks.
	6	blank.
	7 - 12	name of the source node i.
	13 - 18	name of the sink node j.
	19 - 20	n_{ij} (if blank or zero, program sets to 1.)
	21 - 30	c_{ij}
	31 - 40	u_{ij}
	41 - 50	ℓ_{ij}
	51 - 60	Δx_{ij}

The node names identify the ordered pair of nodes corresponding to the arc for which an alteration is desired. The quantity n_{ij} tells the program which of the arcs having this ordered pair of nodes we are trying to alter. If there is only one arc (i,j), columns 19-20 may be left blank, since the program will set n_{ij} to 1. If there is more than one arc (i,j) then n_{ij}th arc (i,j) in the order read will be altered. This arc will have its values for c_{ij}, u_{ij}, and ℓ_{ij} replaced by the values in this card and its value for x_{ij} will be increased by

the value Δx_{ij} in the card. Note that inputting a new x_{ij} is meaningless, since x_{ij} on input is a guess, and guessing a different value of x_{ij} on an alter run would only upset the conservation of flow into the nodes. Hence inputting a non-zero Δx_{ij} is a method for deliberately upsetting the flow conservation.

The effect of ALTER cards is cumulative; the program never returns to the original data. A "save and alter" run may be modified by another "save and alter" run. Figure 3 shows how the cards are set up for a job that begins with a "new problem" run (as it must), is followed by two "save and alter" runs, and is terminated by a "new problem" run. The final control card should be the card "QUIT" as illustrated. The "QUIT" card ends the job.

Figure 3

CONTROL CARD SETUP FOR JOB WITH FOUR RUNS

```
BEGIN                                  }
Title card for Run 1                   }
ARCS                                   }  Run 1
     (arc data cards)                  }
END                                    }
SOLVE                                  }

Title card for Run 2                   }
SAVE                                   }  Run 2
     (alter data cards)                }  (alter run 1)
SOLVE                                  }

Title card for Run 3                   }
SAVE                                   }  Run 3
     (alter data cards)                }  (alter run 2)
SOLVE                                  }

BEGIN                                  }
Title card for Run 4                   }
ARCS                                   }  Run 4
     (arc data cards)                  }  (new run)
END                                    }
SOLVE                                  }

QUIT                                   }  end of job
```

III. THE GENERAL PROBLEM SETUP

The previous section described the use of the program for most purposes. This section describes the program in all of its generality, and this description is arranged so as to correspond to the program subroutines. The reader may wish to skip this section.

The general problem allows one to use reserved tapes, and to make alterations in runs from jobs run previously. The "OUTPUT TAPE" option described below writes the answer on the reserved output tape in a form that can be used subsequently as input. Each time the "OUTPUT TAPE" option is invoked, the program writes what is called a "data package" on the reserved output tape. The record form of a "data package" is as follows:

RECORD FORM OF A DATA PACKAGE

Title card (i.e., record)

ARCS

arc data: one record per arc

NODES

node data: one record per node

END

A data package ends with an "END" record. The node data card is defined later in this section.

Now we will describe the general problem setup. The action of the program is determined by the control cards that it reads. A control card has a specific word punched left-adjusted in columns 1 through 6. The cards are read in four groups, which correspond to four subroutines in the program. Certain controls in each group terminate the group and decide which group is next.

Group 1 (Subroutine PRELIM)

The following controls are read:

BEGIN
SAVE
QUIT
SKIP
TAPE
ARCS
blank

If a card is read that is not one of the above (in cols. 1-6), it is a title card, which is the title printed above the output. A card that is blank in cols 1-6 is a comment and does nothing. The last card of the group is one of the following:

Last Card	Meaning
ARCS	New data to be read, go to group 2 to read arc data.
SAVE	Save the previous run and go immediately to group 4. This is used when one wants to alter a few arcs of the previous run. If a title card is desired it must precede SAVE.
QUIT	This is used at the end of the job. It enables an orderly get-off the machine.

Hence "ARCS" sends program to read group 2 cards, "SAVE" sends program to read group 4 cards, and "QUIT" terminates the job. If "ARCS" is the last card of the group, the run is called a "new problem" run, and if "SAVE" is the last card of the group, the run is called a "save and alter" run. The "ARCS" or "SAVE" card should be preceded by a title card. If a title card is not present, the title of the preceding run will be used on the output. If the first run does not have a title card, the output title may be "hash." Other than the title card and the last card, all cards in group 1 are optional.

The "BEGIN" card ordinarily does nothing -- it is used as the first card of a "new problem" run to give the program a card to look for when the program must pass over a run due to an error. The "BEGIN" card should <u>not</u> be used in a "save and alter" run.

"SKIP" is used in a "new problem" run if one wishes to skip the data for one problem on the <u>reserved input tape</u>. Input is normally from the <u>system input unit</u>. These terms are defined in the section on compiling the program. "SKIP" may be used any number of times or not at all, each "SKIP" causing the reserved input tape to bypass one package of data. The data being skipped is

not interpreted, except that the program recognizes an "END" record as the place to stop skipping.

"TAPE" is used when one desires to begin reading the data from the reserved input tape, after having positioned it, if necessary, with the SKIP card.

In the normal run, neither "SKIP" nor "TAPE" control will be used, and input will continue from the system input unit. The "TAPE" control card causes the reserved input tape to be read and the records on the tape to be processed as control cards until a control card on the reserved tape sends it to group 4, where reading resumes from the system input unit. That is, the reserved tape is read until it reads an "END" card.

The first time an "ARCS" card is read, from the system input unit (or from the reserved tape under "TAPE" control), the program proceeds to group 2.

Group 2 (Subroutine ARCRD)

The only cards in this group are arc data cards as described in the previous section. They are terminated by one of the following cards:

NODES

END

"NODES" sends the program to group 3 and "END" sends the program to group 4.

Group 3 (Subroutine NODERD)

This reads the optional node data cards, which provide the program with a guess as to the final node prices. The format of each data card is:

Col. 1 - 6 blank

7 - 12 name of node

21 - 30 value of node price

The node data cards are terminated by an END control card. Any node that is missing is given a node price of 0. The END card also terminates the group, and sends the program to group 4. If the reserved tape is being read, the program now resumes reading control cards from the system input unit.

Group 4 (Subroutine POSTRD)

This group consists of the following cards:

REFNOD
OUTPUT
ALTER
blank
GO
GOGO
SOLVE

A card that contains punching not in the above set is an (Type 3) error. The cards GO, GOGO, and SOLVE terminate this group and cause the program to go to the SOLVE subroutine, which solves the problem, then to the OUTPUT subroutine, which prints the output, and then back to group 1 for the next run. The other cards in this group may be read in any order. They are all optional. Hence this group need contain only one of the three terminating cards:

GO
GOGO
SOLVE

Which one of the above cards is used to terminate card reading depends upon what data input errors are acceptable. Data input errors are divided into four types, which are described in the next section. Type 1 errors are the least harmful, up to Type 4 errors which are catastrophic. "SOLVE" is used if one wishes to ignore Type 1 and Type 2 errors, "GOGO" is used if one wishes to ignore only Type 1 errors, and "GO" is used if one does not wish to ignore any errors. Hence "SOLVE" is the most permissive and is generally used, except for special problems. The next section on input errors should be read to determine the type of errors one wishes to ignore.

"REFNOD" is an optional data card punched as follows:

Col. 1 - 6 the characters "REFNOD"

7 - 12 the name of the reference node.

If no "REFNOD" card is read, the source node of the first arc will be the reference node. The reference node is the node whose price is not changed by the algorithm. Since a constant may be added to all node prices, we fix one node at its initial value, usually zero, and do not change this value. This provides a measure of uniqueness to the node prices. If there is no "REFNOD" card in a "save and alter" run, the reference node from the previous run is retained.

There are four types of optional "OUTPUT" cards which may be read. These are:

OUTPUT TAPE
OUTPUT PUNCH
OUTPUT NO SYSTEM
OUTPUT IF CUTOFF

Note that these cards have exactly one space between words. Unless otherwise specified, output is always written on the system output unit with carriage control symbols in column 1. The "OUTPUT TAPE" card will cause the output to be written in addition on the reserved output tape in input format. The "OUTPUT PUNCH" card will cause

the output to be in input format using a PUNCH statement. (The node data cards for which the node price is zero are omitted.) The description of what is meant by these tapes is found in the section on compiling the program. If output is not wanted on the system output unit the "OUTPUT NO SYSTEM" card should be used. In addition if the "OUTPUT IF CUTOFF" card is present the output will be made additionally on the reserved output tape if the program is cutoff due to time limitations. The cutoff by time limitation is controlled by a subroutine CUTOFF that must be supplied by the installation. This is described in the section on compiling the program.

The final type of card processed in this group is the ALTER card. The format of this card was given in the previous section. As many ALTER cards (or none) as desired may be read in any run, including a "new problem" run.

OUTPUT options, if desired, must be specified on every run. "REFNOD" need not be specified on a "save and alter" run.

This ends the description of group 4 cards, and hence of all cards. Table 1 summarizes the control cards in the four groups. Note that in our terminology data cards are control cards (indeed all cards are control cards) but some control cards are not data cards.

Table 1

Summary of Control Card Input

	Group 1	Group 2	Group 3	Group 4
Subroutine Name	PRELIM	ARCRD	NODERD	POSTRD
Control Cards Recognized	BEGIN SAVE QUIT SKIP TAPE blank title ARCS	blank NODES END	blank END	REFNOD OUTPUT blank ALTER GO GOGO SOLVE
Data Cards Recognized	none	blank (arc data)	blank (node data)	REFNOD (ref. node) ALTER [≠blank] (alter data)
Control Cards which terminate the subroutine	SAVE ARCS QUIT	NODES END	END	GO GOGO SOLVE
What does the program do when this subroutine is terminated?	Go to Group 2 (if ARCS) or Go to Group 4 (if SAVE) or Get off the machine (if QUIT)	Go to Group 3 (if NODES) or Go to Group 4 (if END)	Go to Group 4	Solve the problem; output; then go to Group 1
Are control cards printed on system output unit when read?	Yes	No	No	Yes

IV. INPUT ERROR MESSAGES

Input error messages are distinguished by one or more asterisks (*) in the message. The number of asterisks in the messages indicate the severity of the error (See Table 2). As explained in the preceding section, the program will ignore errors of severity 1 or 2 if the "SOLVE" control card is used, and it will ignore errors of severity 1 if the control card "GOGO" is used. If the control card "GO" is used, no errors will be ignored. Error messages are printed even though they are ignored.

Type 1 Errors

Message #10, #11, #12 (Table 2) occur when the network is a transportation problem. In a transportation problem some nodes may be only sources for arcs (Message #11) and other nodes may only be sinks for arcs (Message #12). Nodes may have a net flow into them. The signed number in Message #10 is the sum of the flows into the node as given by the initial values for x_{ij}.

Message #6, printed when a node card is read for a node that does not exist in the arcs, is also considered to be a trivial error, since the initial node price does not affect the validity of the solution.

Table 2

Input Error Messages

Message #	Error Type	Message
1	4	Too many arcs in this run ****
2	4	Too many nodes in this run ****
3	2	** Source nodes not adjacent, arc A B
4	3	*** Field error in arc number 17
5	2	** Field error in node card number 19
6	1	* Card 9, Node B not in arcs
7	3	*** Illegal card =
8	3	*** Arc on above ALTER card not defined
9	2	** Arc C D has lower bound greater than upper
10	1	* Node E non-conservative, net flow = -2
11	1	* No arc ends at node ABC
12	1	* No arc begins at node DEF
13	4	**** No arc data in this run

Type 2 Errors

Type 2 errors are actual input errors, but are of the type that the program can correct. If the arcs are out of order (Message #3) the program re-arranges them; if the lower bound exceeds the upper bound (Message #9) the program exchanges them; an error in the node price cards (Message #5) is ignored.

Type 3 Errors

Type 3 errors are "field" errors; that is, there are punches in the first twenty columns that the program cannot interpret. The problem will not be solved, but card reading will continue until the end of the control cards for the current run. Then the program will pass over all cards until it reaches a "BEGIN" card, from where it will commence processing control cards as usual.

A field error in an arc card (Message #4) is a punch in columns 1 through 6 that is neither all blank nor "END" nor "NODES." Message #7 denotes a field error in a Group 4 control card -- the first 18 columns of this card are printed after the equal sign. Message #8 is printed when the altered arc was not present in memory. Only arcs that were read in the previous data may be altered. Furthermore there must be at least n_{ij} arcs of type (i,j) in memory.

Type 4 Errors

Type 4 errors (Messages #1, #2) occur when the problem is too big for memory. See the section on compiling the program for setting up dimensions of arrays. At the occurrence of this type of error, the program passes over control cards until it reaches a "BEGIN" or "QUIT" control. It then begins normal processing with this card. Message #13 occurs if there are no data cards following the "ARCS" card or if "SAVE" is the first run of a job. In this case the problem is too small (no arcs!).

V. OUTPUT

The output generated by the program is, for the most part, self-explanatory. Figure 2 shows the output for the arcs for a feasible problem. A list of the node prices is produced after the above arc output. If the problem is cutoff due to time limitations or if the problem is infeasible, no "K" is printed at the end of the line; instead an "N" is printed for arcs out-of-kilter and nothing is printed for arcs in kilter.

When the program encounters an arc that cannot be brought into kilter, the program proceeds to the next arc.

A count is kept of the number of arcs that could not be brought into kilter. If this number is non-zero (which is true only if the problem is infeasible), this number is printed before the arc output, viz.;

3 ARCS OUT OF KILTER

Then, the program goes back to the first arc that could not be brought into kilter and determines which nodes were labeled. These nodes are designated by an * after the node price. In addition, the word "CUT" is printed beside each arc that has its sink node labeled and its source node unlabeled, and the word "CUT*" is printed beside each arc that has its source node labeled and its sink node unlabeled. These words give the user a clue as to which arcs to change to make the problem feasible. Either an arc designated "CUT*" must have its upper bound raised and/or an arc designated "CUT" must have its lower bound decreased.

VI. COMPILING THE PROGRAM

The deck submitted to SHARE is a FORTRAN IV source deck punched in EBCDIC with control cards for the IBM System 370 Model 145.

It is suggested that the procedure to be described be followed when first using this program. First, determine the number of the system input unit and the number of the

system output unit at your installation and change, if necessary, the values of KI and KO as set at the beginning of the main program to agree with your installation.

In the program supplied,

System input unit = KI = 5

System output unit = KO = 6.

Next, supply the "JOB" card at the beginning of the deck as required by your installation. Now, you may run this deck, including the test deck at the end, on your machine. The result of the first problem should agree with Figure 2.

Reserved Tapes

There are optional control cards in the general program that allow the use of reserved tapes. The units that correspond to these reserved tapes are set by statements in the main program. The reserved input tape in the SHARE deck is unit 9 as set by the statement

```
KQ(3) = 9
```

The reserved output tape is unit 10 as set by the statement

```
KQ(2) = 10
```

If other units are desired, the definition of KQ(3) and/or KQ(2) should be changed to the numbers desired. If reserved tapes are not used, the values of KQ(3) and KQ(2) are irrelevant.

Infinity

The algorithm makes use of "infinity" in the sense that "infinity" must be larger than any cost or flow value used. Infinity is set in the main program by the statement

IFIN = 2147483647.

This is the largest integer representable in a 32-bit machine. For most problems this number is sufficiently large, but on machines with a word size larger than 32-bits, this number may be increased up to the largest integer representable.

Dimensions

The SHARE deck is set up to run problems with no more than 1500 nodes and 3000 arcs. Dimensions are set in a COMMON statement (3 cards) near the beginning of most of the subprograms. These dimensions may be increased by an amount that is dependent on the type and size of the machine being used. Let NODMAX be the maximum number of nodes desired and ARCMAX the maximum number of arcs desired. Then the symbols NL, NN, NP and IJ in the COMMON statements should be dimensioned NODMAX + 1, and the symbols JI, KC, KU, KX, JA, IB and LW should be dimensioned ARCMAX. In addition, the statements in the main program which define KQ(4) and KQ(5) must be changed to read:

KQ(4) = ARCMAX

KQ(5) = NODMAX

Boundary Alignment for 370 COMMON Statement

The 370 requires that words longer than one byte begin at a byte location which is a multiple of the number of bytes in the word. This alignment is correct in the SHARE deck COMMON statement. The alignment will remain correct if NODMAX and ARCMAX are even numbers.

COMMON Size on System/370

The number of bytes used by the COMMON statement (without small integers) is

28*NODMAX+28*ARCMAX+348.

In the SHARE deck, this is 42,348 (decimal) bytes.

If the INTEGER*2) statement is used, the size is

22*NODMAX+22*ARCMAX+344.

Time Limitation Cutoff

The program calls SUBROUTINE CUTOFF to determine whether time is running out. CUTOFF sets I = 1 if time is not running out, and sets I to any other number if time is running out. The value for I is always 1 in the version supplied. The statement

```
I = 1
```

should be replaced by the instruction or instructions which are used at your installation for time cutoff. For example, if sense switch 5 is used for cutoff, use the instruction

```
CALL SSWTCH (5,I).
```

Overflow

It is possible for the node prices to overflow in SUBROUTINE RAISE. If a fixed point overflow test is available on your machine, set

LER = -3

near the end of the subroutine, as indicated by the comments, if there is an overflow. If an overflow occurs that is not caught by the program, it will be indicated by "funny" numbers appearing in the cost column of the output.

Short Integers

Space may be saved on the 370 by using the following card in every subprogram that has a COMMON statement:

```
INTEGER*2 IJ,IL,JL,JI,JA,IB
```

These symbols are used for arc and node numbers. The short integers will be long enough, provided the problems have no more than 32,766 arcs or nodes. It is possible to solve problems this large on machines of one-million bytes.

Compiler Versions

Some 370 systems have more than one FORTRAN IV compiler. Use the version that produces the most efficient object program.

Re-Coding

Computing time can be saved by re-coding certain subroutines in machine language, or MAP, PL/1, etc., depending

on the machine being used. More than half of the execution time is usually spent in the LABEL subroutine. Index register optimization can generally be more efficiently done "by hand" than by the FORTRAN compiler. Subroutines BREAKT and RAISE might also be substantially improved by re-coding.

VII. THE NETWORK SUBROUTINE

The subprograms SOLVE, KILTER, LABEL, BREAKT, RAISE, and CUTOFF may be used as a network subroutine. This can be done when a solution of a network is needed as a part of another problem. In order to use these subprograms as a network subroutine, the data must be set up so that arcs with the same source nodes are adjacent. When this is done, number the nodes in the order in which they appear as source nodes. For example, in Figure 1, node 1 is "SM," node 2 is "LA," ..., node 22 is "VCTRVL." If there are any nodes remaining (that do not appear as source nodes), continue numbering consecutively any remaining nodes. The total number of nodes is m. The arcs are numbered from 1 through n. Then set up the following arrays:

KC(1) = c_1	LW(1) = ℓ_1	KU(1) = u_1	KX(1) = x_1
KC(2) = c_2	LW(2) = ℓ_2	KU(2) = u_2	KX(2) = x_2
.	.	.	.
.	.	.	.
.	.	.	.
KC(n) = c_n	LW(n) = ℓ_n	KU(n) = u_n	KX(n) = x_n

where c_j is the unit cost for arc j; ℓ_j is lower capacity; u_j is upper capacity; x_j is the nominal (or initial) flow.

Also: JA(1) = node number of sink node for arc 1.

JA(2) = node number of sink node for arc 2.

.
.
.

JA(n) = node number of sink node for arc n.

.

NP(1) = initial node price for node 1

NP(2) = initial node price for node 2

.
.
.

NP(m) = initial node price for node m.

The initial node prices are usually zero.

Also: IL(1) = number of first arc whose source node is 1 (=1),

IL(2) = number of first arc whose source node is 2,

.
.
.

IL(m) = number of first arc whose source node is m,

IL(m+1) = n+1.

If node j does not appear as a source node then IL(j) = n+1. Note that IL(m+1) must not be omitted or the program will loop. For Figure 1, IL(1) = 1, IL(2) = 4, IL(3) = 8, etc. Note also that successive IL values must not decrease.

Finally set the following values:

LC(5) = 0
LC(6) = 0
LC(7) = 0
LC(8) = 0
M = number of nodes = m
N = number of arcs = n
LER = 0
KAT = 1 (or number of reference node)
KO = device number of the system output unit
KQ(8) = 2
IFIN = 999999999 (or some large number)

The output device number is used for error messages in SUBROUTINE SOLVE. If messages (for errors #9, #10, #11, Table 2) are not wanted, remove all of the "WRITE" statements in this subroutine. If messages are wanted, the node names should be in

NN(1) = Name of node 1
NN(2) = Name of node 2
.
.
.
NN(m) = Name of node m.

The COMMON statements (3 cards) in SOLVE, KILTER, LABEL, BREAKT and RAISE should be identical, and their COMMON statement must also appear in the program which sets up the above arrays. To execute the network subroutine, simply,

CALL SOLVE(KE).

When the subroutine returns control to the calling program, KX(1), KX(2), ..., KX(n) will contain an optimal solution if

LER = 0

If LER = -1, the problem is infeasible, and KE has been set to the number of arcs out-of-kilter. If LER = -2, the algorithm was terminated prematurely because time ran out (as set by SUBROUTINE CUTOFF; see description of this subroutine elsewhere). Other results returned are the optimal node prices in NP(1) through NP(m); LC(5), the number of breakthroughs, LC(6); the number of non-breakthroughs, LC(7); the number of flow changes made; and LC(8), the total number of scans made for labeling.

The rules for changing the COMMON statement are the same as given in the section on compiling the program. The COMMON statement must appear in every subprogram that uses any of the above symbols. Also, any specification statement (such as REAL*8, INTEGER*2) that refer to COMMON variables must appear with every COMMON statement. The COMMON statement may be given a label if desired.