CANADA

DEPARTMENT OF COMMUNICATIONS

OTTAWA

NETWORKS AND SYSTEM STUDIES

John deMercado
Roger Robert
Nicholas Spyratos
Kalman Toth

TERRESTRIAL PLANNING BRANCH

SUMMER PROGRAM 1971

DEPARTMENT OF COMMUNICATIONS

CANADA

MINISTÈRE DES COMMUNICATIONS

FOREWORD

This work represents the Summer Student Program in the Terrestrial Planning Branch.

Mr. Roger Robert wrote the program MAT(,PROB) for Matrix and Reliability Analysis. This program provided the computation for the analytical results in a paper written at the same time by John deMercado, "Reliability Prediction Technique For System With Many Failed States" which will be published in the IEEE Transaction on Reliability Theory in November 1971. Mr. Robert also wrote the program MAX(,MIN) which provides various algorithms for computing optimal decision making methods originally developed by Ronald Howard at MIT in 1960.

Mr. Kalman Toth wrote the three network synthesis programs, SHORT (1), SHORT (2), and NET(SYM). The theoretical details of the first two constitute his master's thesis to be submitted to Carleton University this fall. The theoretical details of NET(SYM) can be found in a paper (not attached) "On The Synthesis of Non Flow Redundant Networks" by John deMercado and Nicholas Spyratos which will be presented at the Computer-Communication network meeting at Brooklyn Polytechnic in April 1972 and ultimately printed as a chapter in the Book of the Proceedings. Program NET PLAN was developed under contract by the Faculty of Management Sciences - Ottawa University.

Dr. J. deMercado,
Director.

TABLE OF CONTENTS

SUMMARY OF PROGRAMMING PACKAGES
AVAILABLE FROM
TERRESTRIAL PLANNING BRANCH

PROGRAM NAME	PROGRAM DESCRIPTION
SHORT 1(,K)	THIS PROGRAM SYNTHESIZES FROM GIVEN TERMINAL REQUIREMENTS AND ARC COSTS, A COMMUNICATIONS NETWORK IN WHICH COMMUNICATION BETWEEN ALL PAIRS OF NODES EXISTS AT THE SAME TIME (SIMULTANEOUS TRANSMISSION). TOTAL NETWORK COST IS MINIMIZED.
SHORT 2(,K)	SAME AS ABOVE EXCEPT THAT ONLY ONE PAIR OF TERMINALS COMMUNICATES AT ONE TIME (TIME-SHARED COMMUNICATIONS). TOTAL NETWORK COST IS REDUCED.
NETPLAN(,K)	GIVEN A COMMUNICATIONS NETWORK WITH ARC CAPABILITIES, ARC RENTAL COSTS AND ARC EXPANSION COSTS ALSO GIVEN FOR A GIVEN PAIR OF NODES, THIS PROGRAM COMPUTES THE OPTIMAL NEW FLOW PATTERNS AND THE OPTIMAL EXPANSION PATTERNS.
NETSYM 1(,K)	GIVEN THE TERMINAL REQUIREMENTS AND THE ARC CONSTRAINTS A TIME-SHARED COMMUNICATIONS, THIS PROGRAM SYNTHESIZES A NETWORK IN WHICH ALL TERMINAL REQUIREMENTS ARE EXACTLY SATISFIED.
MAT(,PROB)	THIS PROGRAM HAS A NUMBER OF VARIOUS MATRIX OPERATIONS, SUCH AS MATRIX INVERSION, FUNCTIONS OF A MATRIX, ETC. IT ALSO PERFORMS VARIOUS SYSTEM RELIABILITY SIMULATIONS.
MAX(,MIN)	THIS PROGRAM ALLOWS OPTIMAL STRATERGIES TO BE OBTAINED WITH CORRESPONDING MAXIMUM GAIN OR MINIMUM LOSS, FOR VARIOUS DECISION MAKING PROBLEMS.

SECTION 1

System Reliability Modelling & Simulation Package

by

John deMercado & Roger Robert

Department of Communications

Ottawa

September 1971

TABLE OF CONTENTS

Introduction

This report is divided into two parts, PART I, describes the use of a "conversational" computer program that has been developed by Roger Robert. This computer program implements the analytical results for systems reliability modelling presented in a recent paper. *) The program has been written to be "conversational" in either the French or English language, and is available to departmental users at headquarters and C.R.C. on a time shared basis from the Sigma 7 at Shirley's Bay.

The program in Part I also includes various options, for computing various operations on matrices, such as inversion, addition, and various functions of matrices including exponential functions.

For further details of the theoretical considerations involved, the reader should consult the paper listed below, a copy of which is attached as an Appendix to Part I.

Included in this report are a list of the current telephone numbers (which could change with time) for various speed lines to the Sigma 7 and a number of examples showing in detail the formulation of the reliability problem and the data input and output formats.

---

*) Reliability Prediction Techniques for Systems With Many Failed States by John deMercado, (to appear) in the IEEE transactions on Reliability Theory, November 1971

The program in Part II, is based on the work of Howard *) who solved the Sequential Decision Problem, by a combination of techniques from the theory of Marko Processes and Dynamic Programming. This program therefore implements How ard's Analytical expressions for finding the optimal strategy and corresponding maximum reward function or minimum loss function for a decision making problem that can be modelled by a Markov chain having known transition probability matrix [P] and corresponding reward/strategy matrix [R].

OBTAINING ACCESS TO THE COMPUTER (AT SHIRLEY'S BAY)

The following telephone ports are currently available for "dial up terminal" usage of the Sigma 7. Contact with the computer can be made through any of the numbers as listed in Table I below:

TABLE I

DIAL	SPEED	TIME
9-828-2754	10 characters/second	any-time
996-7051 Ext. 505,506 507 or 508	30 characters/second	day only
996-6723 or 996-6724 or 996-6725	30 characters/second	night only

*) Dynamic Programming and Markov Processes, by R.A. Howard, MIT press. 1960.

If an acoustically coupled terminal is being used, the user must first ensure that the computer has acknowledged contact before placing the telephone receiver in the acoustic couple. A high pitched screeching sound in the telephone receiver indicates acknowledgement of contact. Details of the options available on the Sigma 7 are available in the "XDS Systems Manual Supplement".

PROCEDURE FOR USE OF THE PROGRAMS (PART I or PART II).

The following procedure will enable the program to be accessed. First of all, dial the computer and establish contact. The computer will then begin the initial dialogue shown below with the user who should respond by typing in the statements as underlined on the right below.

COMPUTER DIALOGUE FORMAT - PART I

(note ↵, means the user should press the carriage return key on the teletype)

BTM SYSTEM IS UP.

(Date), (time)

! LOGIN : PLANNING 1004S POLICY

! BA

> PAS PROB ↵

> LOAD MAT ↵

> FAS ↵

General Instructions

The computer will then ask whether or not the user wishes to work in English or French and the user response should be made by entering "E" or "F" as appropriate.

Regardless of which option is chosen from Table II or Table III, all typewritten input should follow the following four guidelines:

(1) If a single letter or word is to be typed in answer to some query from the computer, it should be typed in quotation marks ("   ").

(2) The values of the matrices being entered should always be typed by row and be separated by commas.

(3) At the end of a line of type, it is necessary to press the carriage return (indicated above by ↓) in order to continue the input or simply come to another line.

(4) At the end of program use, the user should sign off by typing "BYE".

PART I OPTIONS

The program of Part I has been written to accept any matrix up to a size of 24 x 24. The following are available once contact with the main program has been made as indicated above.

Matrix Function Options

TABLE II

OPTION	CODE	DESCRIPTION OF OPTION	Input required in addition to Code
1	AEX	Raises a matrix [A] to the powers 1 to N That is computer $[A]^k$, for k = 1, ---N.	a) Dimension of matrix b) Entries of matrix typed in by row c) Maximum value of N
2	AEX1	Computers $[A]^k$, for one value, namely k = N	SAME AS ABOVE
3	ADD	Addition of two matrices [A] and [B]	a) Dimension of matrices b) Entries in the matrices by row.
4	SST	Substraction of two matrices [A] and [B]	SAME AS ABOVE
5	INO	Inversion of Matrix [A]	a) Dimension of matrix b) Entries of matrix typed in by row.
6	INF	Inversion of matrix $\{[I] - \varkappa [A]\}$ [I] is identity matrix, [A] square matrix and $\varkappa$ a scaler	a) Dimension of matrix [A] b) Value of $\varkappa$ c) Entries of matrix [A] by row.
7	EEM	Computes the function: exponential ([A] k), for k = 1, ---N..	a) Dimension of matrix [A] b) Entries of matrix by row c) Value of N.
8	EEM1	Computes exponential ([A] k) for the value k = N only.	SAME AS ABOVE

PART I - OPTIONS (Cont'd).

Systems Reliability Modelling Options *)

OPTION	CODE	DESCRIPTION	Input required in addition to code.
9	STF	Computes the Steady State Failure Probability Matrix [P], where $[P] = [[I] - [A]][B]$	a) Dimensions [A] and [B] b) Entries of [A] and [B] by row.
10	TRP	Computes the Steady State Probability Failure Functions $[P(N)]$ where $[P(N)] = [A]^N[B] + [P(N-1)]$	a) Dimensions of [A] and [B] by row b) Entries [A] and [B] by row c) Value of N.
11	RF	Computes the Reliability Function R(N), N to M for the system having K acceptable states. Where R(N) = S(N)j and S(N)j. j = 1 are the entries of the Vectors $\bar{S}(N) = \bar{S}(0)[A]$	a) Number of states R b) Values of $\bar{S}(0)$ c) Entries of matrix [A] d) Time "N" to "M"

TABLE III

The following example illustrates the approach to the reliability modelling of a) systems and b) networks and the procedure for obtaining the matrices [A] and [B] . Copies of typical printout for a simulation of this problem is included with the example.

---

*) See the paper cited in the introduction for theoretical details.

EXAMPLE - RELIABILITY ANALYSIS

Example 1

Consider the following portion of a telecommunication network (Figure 1). The state assignment that describes the operational aspects that we are interested in for this network is shown in Figure 2.

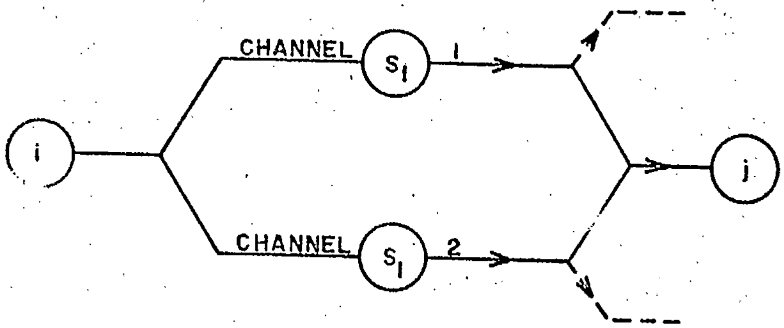


FIGURE 1

State Assignment

State	Word Description
A_1	Both S_1 and S_2 provide a path from i to j
A_2	S_1 fails and the only path is provided by S_2. Repairs to S_1 are not yet started.
A_3	Repairs to S_1 start. S_2 is still providing the connections between i and j.
F_1	S_2 fails before repairs to S_1 have begun.
F_2	S_2 fails before repairs to S_1 are completed.

Figure 2

The transition graph, drawn from Figure 2 is shown below in Figure 3.

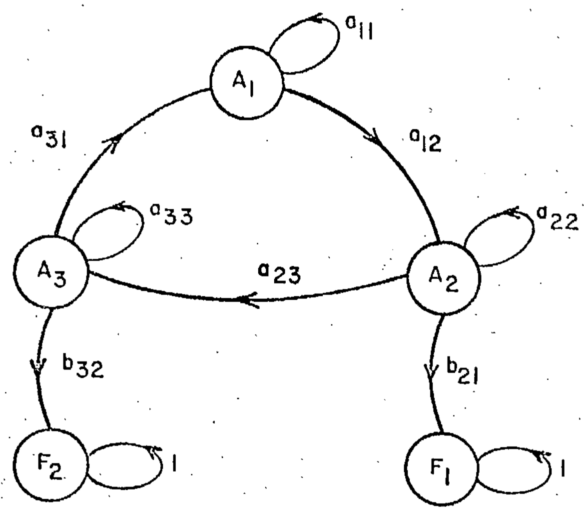


FIGURE 3

For computational purposes we take the failure, repair and delay-repair rates, λ, μ, ρ to be $\lambda = .002/\text{hr.}$ $\mu = .004/\text{hour}$ $\rho = .2/\text{hr.}$ The $|M|$ matrix for this example is therefore obtained as shown in Figure 4 .

$|M| \equiv$

	A_1	A_2	A_3	F_1	F_2
A_1	.998	.002	0	0	0
A_2	0	.798	.2	.002	0
A_3	.004	0	.994	0	.002
F_1	0	0	0	1	0
F_2	0	0	0	0	1

$= \left[\begin{array}{c|c} [A] & [B] \\ \hline [O] & [I] \end{array}\right]$

Figure 4

A computer simulation [15], for $|P(10)|$, $|P(50)|$, $|P|$ as well as the reliability function R(n) for three different initial state vectors $\bar{s}(o)$, is shown in Figure 5.

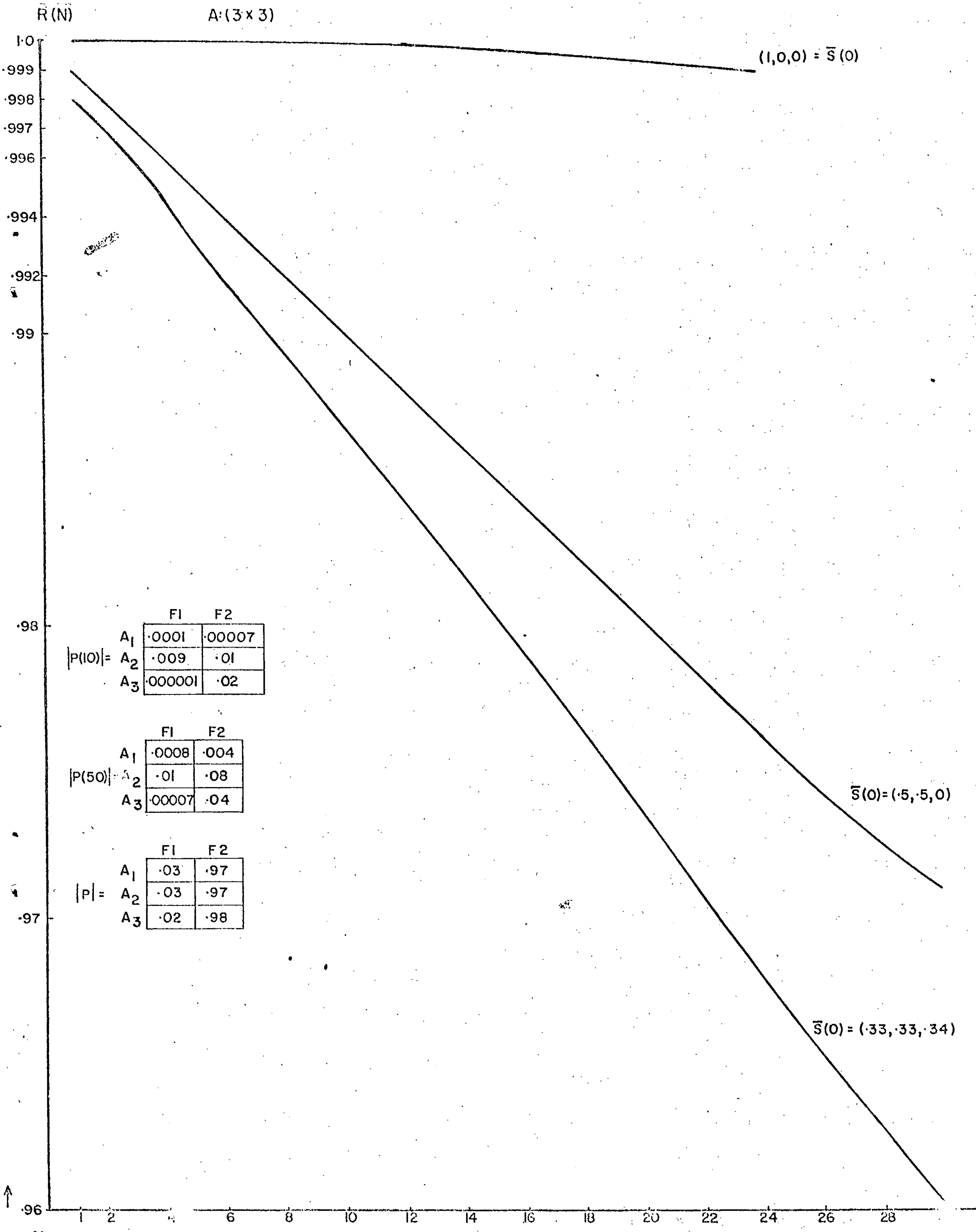

R̄(N)
A:(3×3)
1·0
·999
·998
·997
·996
·994
·992
·99
·98
·97
·96
(1,0,0) = S̄(0)
S̄(0)=(·5,·5,0)
S̄(0) = (·33,·33,·34)
|P(10)| =
F1 F2
A1 ·0001 ·00007
A2 ·009 ·01
A3 ·000001 ·02
|P(50)| =
F1 F2
A1 ·0008 ·004
A2 ·01 ·08
A3 ·00007 ·04
|P| =
F1 F2
A1 ·03 ·97
A2 ·03 ·97
A3 ·02 ·98
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28
N=

FIGURE 5

```
!~
BTM SYSTEM-C IS UP
09/15/'71 15:31
!LOGIN: PLANNING,1004S,POLICY
ID= 3
!BASIC
6PASPROB
>LOADMAT
>FAS
15:32   09/15

TYPE #E# IF YOU WISH TO WORK IN ENGLISH, #F# FOR FRENCH
TAPEZ #F# SI VOUS DESIREZ TRAVAILLER EN FRANCAIS,ET #E# EN ANGLAIS

 N.B. VOUS DEVEZ TAPER CETTE LETTRE ENTRE GUILLEMETS.
 N.B. YOU MUST TYPE THIS LETTER BETWEEN DOUBLE QUOTATIONS.

?"E"
DO YOU WISH A DEFINITION OF OPERATION CODES? (#Y#,#N#)
?"N"
ENTER OPERATION CODE (BETWEEN DOUBLE QUOTATIONS)
?"TRP"
ENTER SIZE OF YOUR (FIRST) MATRIX (ROWS,COLUMNS)
?3,3
ENTER THE #A# MATRIX IN: P(K)=(A(EXP.(K-1))*B)+P(K-1)
?.998,.002,0,0,.798,.2,.004,0,.994
ENTER THE NUMBER OF COLUMNS IN THE #B# MATRIX
?2
ENTER MATRIX #B#
?0,0,.002,0,0,.002
```

```
DO YOU WISH RESULTS FOR P(1) TO P(N)? (#Y# OR #N#)
="N"
FOR HOW MANY VALUES DO YOU WISH RESULTS?
53
ENTER THESE VALUES
?1,10,50

P( 1)=

 0              0

 2.00000E-03    0
÷
  0               2.00000E-03

P( 10)=

 1.09569E-04    6.79955E-05

 8.86457E-03    1.08209E-02

 1.35991E-06    1.94691E-02

P( 50)=

 8.55040E-04    3.53820E-03

 9.95763E-03    7.84276E-02

 7.07641E-05    8.68167E-02

DO YOU WISH #STF# RESULTS FOR SAME VALUES?
?"Y"

UTEADY STATE FAILURE PROBABILITIES

P=

 2.91262E-02    .970874

 2.91262E-02    .970874

 1.94175E-02    .980583
```

```
DO YOU WISH CALCULATIONS FOR OTHER MATRICES? (#Y#,#N#)
?"Y"
ENTER OPERATION CODE (BETWEEN DOUBLE QUOTATIONS)
?"RF"
ENTER THE NUMBER OF COMPONENTS IN THE S(0) VECTOR
=3
ENTER MATRIX #A#
?.998,.002,0,0,.798,.2,.004,0,.994
ENTER S(0) VECTOR
?0,0←←←.5,.5,0
ENTER #N# AND #M#, THE FUNCTION IS GIVEN FROM TIME #N# TO TIME #M#
=1,20

RELIABILITY FUNCTION


INITIAL PROBABILITY FUNCTION=
 .500000        .500000       0
```

N=	1	2	3	4
R(N)	.999000	.998000	.997001	.996003
N=	5	6	7	8
R(N)	.995007	.994014	.993023	.992035
N=	9	10	11	12
R(N)	.991050	.990068	.989090	.988115
N=	13	14	15	16
R(N)	.987144	.986176	.985212	.984251
N=	17	18	19	20
R(N)	.983293	.982340	.981390	.980443

```
DO YOU WISH THE GRAPH OF THIS FUNCTION? (#Y# OR #N#)
?"N"
DO JOU WISH THE SAME CALCULATIONS FOR ANOTHER S(0)? (#Y# OR #N#)
?"N"
DO YOU WISH CALCULATIONS FOR OTHER MATRICES? (#Y#,#N#)
?"N"

12150 HALT
>
```

PART II

DECISION MAKING

Computer Dialogue Format - Part II

Once contact is established with the Computer, the dialogue begins as indicated below. The user types the underlined responses at the right in answer to the computer queries or statements (not underlined).

BTM SYSTEM IS UP
(Date) , (Time)
! LOGIN : PLANNING, 10045, POLICY
! BA
> PAS MIN
> LOAD MAX
> FAS

GENERAL INSTRUCTIONS

These are the same as in Part I, however, the options for this package are as described in Table IV below.

PART II - OPTIONS

These six options are concerned with determining the optimum strategies and associated minimum loss or maximum reward function for a decision making problem that can be modelled by a N state markov chain with known transition probabilities [P] and one step transition rewards*) [R].

OPTIONS	CODE	DESCRIPTION	Input Required in addition to Code.
1	1	Maximize Reward Function and find Optimum Strategies	a) Number of states (max. 10) b) No. of alternatives for each state c) Matrices [P] and [R] d) Maximum time N (max.25).
2	2	Minimize Loss Function and find Optimum Strategies	SAME AS ABOVE
3	3	Maximize Reward Function for Constant Discounting Environment and find Optimum Strategies	SAME AS ABOVE plus Discount constant C
4	4	Maximize Reward Function in variable Discounting Environment and find Optimum Strategies	SAME AS OPTION (1) plus Discount vectors $\overline{C(N)}$
5	5	Minimize Loss Function in Constant Discounting Environment and Find Optimum Strategies	SAME AS OPTION (3)
6	6	Minimize Loss Function in Variable Discounting Environment and Find Optimum Strategies	SAME AS OPTION (4)

TABLE IV

*) For further details, See Howard in particular chapters 3-6.

In all cases the output from these options is of the form

N –	0	1	2	3	– – – –	25
$V_1(N)$ =	*	*				*
$V_i(N)$ =	*	*				*
$D_1(N)$ =	*	*				*
$D_i(N)$ =	*	*				*

Where $V_i(N)$, $i = 1, -- M$, is the maximum reward or minimum loss in state i at time N and the corresponding decision at that time is $D_i(N)$. It should be noted that the decision making process does converge. That is, there are a optimum set of strategies $\overline{D}(N)$ to follow, so that the changes in the Vector $\overline{V}(N)$ in successive instants of time will be constant and maximum or minimum as the case may be.

The following is the computer simulation of the example given in Howard pages 28 and 29 (also attached).

EXAMPLE

DECISION MAKING

The Toymaker's Problem

The alternatives for the toymaker are presented in Table 3.1. The quantity q_i^k is the expected reward from a single transition from state i under alternative k. Thus, $q_i^k = \sum_{j=1}^{N} p_{ij}^k r_{ij}^k$.

Table 3.1. THE TOYMAKER'S SEQUENTIAL DECISION PROBLEM

State i	Alternative k	Transition Probabilities p_{i1}^k	p_{i2}^k	Rewards r_{i1}^k	r_{i2}^k	Expected Immediate Reward q_i^k
1 (Successful toy)	1 (No advertising)	0.5	0.5	9	3	6
	2 (Advertising)	0.8	0.2	4	4	4
2 (Unsuccessful toy)	1 (No research)	0.4	0.6	3	−7	−3
	2 (Research)	0.7	0.3	1	−19	−5

Suppose that the toymaker has n weeks remaining before his business will close down. We shall call n the number of stages remaining in the process. The toymaker would like to know as a function of n and his present state what alternative he should use for the next transition (week) in order to maximize the total earnings of his business over the n-week period.

We shall define $d_i(n)$ as the number of the alternative in the ith state that will be used at stage n. We call $d_i(n)$ the "decision" in state i at the nth stage. When $d_i(n)$ has been specified for all i and all n, a "policy" has been determined. The optimal policy is the one that maximizes total expected return for each i and n.

To analyze this problem, let us redefine $v_i(n)$ as the total expected return in n stages starting from state i *if an optimal policy is followed.* It follows that for any n

$$v_i(n+1) = \max_k \sum_{j=1}^{N} p_{ij}^k[r_{ij}^k + v_j(n)] \qquad n = 0, 1, 2, \cdots \tag{3.1}$$

Suppose that we have decided which alternatives to follow at stages n, $n-1, \cdots, 1$ in such a way that we have maximized $v_j(n)$ for $j = 1, 2, \cdots, N$. We are at stage $n+1$ and are seeking the alternative we should follow in the ith state in order to make $v_i(n+1)$ as large as possible; this is $d_i(n+1)$. If we used alternative k in the ith state, then our expected return for $n+1$ stages would be

$$\sum_{j=1}^{N} p_{ij}^k[r_{ij}^k + v_j(n)] \tag{3.2}$$

by the argument of Chapter 2. We are seeking the alternative in the ith state that will maximize Expression 3.2. For this alternative, $v_i(n+1)$ will be equal to Expression 3.2; thus we have derived Eq. 3.1,* which we may call the value iteration equation. Equation 3.1 may be written in terms of the expected immediate rewards from each alternative in the form

$$v_i(n+1) = \max_k \left[q_i^k + \sum_{j=1}^{N} p_{ij}^k v_j(n)\right] \tag{3.3}$$

The use of the recursive relation (Eq. 3.3) will tell the toymaker which alternative to use in each state at each stage and will also provide him with his expected future earnings at each stage of the process. To apply this relation, we must specify $v_j(0)$ the boundary condition for the process. We shall assign the value 0 to both $v_1(0)$ and $v_2(0)$, as we did in Chapter 2. Now Eq. 3.3 will be used to solve the toymaker's problem as presented in Table 3.1. The results are shown in Table 3.2.

Table 3.2. TOYMAKER'S PROBLEM SOLVED BY VALUE ITERATION

$n =$	0	1	2	3	4	$\cdots$
$v_1(n)$	0	6	8.2	10.22	12.222	$\cdots$
$v_2(n)$	0	-3	-1.7	0.23	2.223	$\cdots$
$d_1(n)$	—	1	2	2	2	$\cdots$
$d_2(n)$	—	1	2	2	2	$\cdots$

The calculation will be illustrated by finding the alternatives and

```
 SYS

!BASIC
>PASMIN
>LOADMAX
>PAS
15:45   09/15
TYPE #E# IF YOU WISH TO WORK IN ENGLISH,#F# FOR FRENCH
TAPEZ #F# SI VOUS DESIREZ TRAVAILLER EN FRANCAIS,ET #E# EN ANGLAIS

 N.B. VOUS DEVEZ TAPER CETTE LETTRE ENTRE GUILLEMETS!
 N.B. YOU MUST TYPE THIS LETTER BETWEEN DOUBLE QUOTATIONS!

EN CAS D'ERREUR, VOUS OBTIENDREZ UN MESSAGE APPROPRIE,
      VOUS POURREZ ALORS TAPER LES BONS CARACTERES.
IF YOU MAKE A MISTAKE,YOU'LL GET A MESSAGE OF ERROR,
        JUST RETYPE CORRECTLY.
?"E"

TYPE #Y# IF YOU WISH A DESCRIPTION OF THE OPTIONS, TYPE #N# IF NOT.
?"N"
ENTER THE MAXIMUM TIME,#N#, AND THE NUMBER OF STATES,#M#
?11,2
ENTER THE NUMBER OF ALTERNATIVES FOR EACH STATE
;2,2
ENTER EACH #P# MATRIX, BY ROW
;.5,.5,.8,.2,.4,.6,.7,.3
ENTER EACH #R# MATRIX, BY ROW
?9,3,4,4,3,-7,1,-19
ENTER V(0) VECTOR
?0,0
DO YOU WISH A DISPLAY OF THE INPUT DATA? (#Y# OR #N#)
>"N"
```

```
WHICH OPTION DO YOU WISH? (1,2,3,4,5,6)
?1


OPTION 1

N=                 0             1             2             3
----------------------------------------------------------------------

V 1(N)             0             6             8.20000       10.2200
V 2(N)             0            -3            -1.70000       .230000

D 1(N)             -             1             2             2
D 2(N)             -             1             2             2

        ----                  ----                  ----

N=                 4             5             6             7
----------------------------------------------------------------------

V 1(N)             12.2220       14.2222       16.2222       18.2222
V 2(N)             2.22300       4.22230       6.22223       8.22222

D 1(N)             2             2             2             2
D 2(N)             2             2             2             2

        ----                  ----                  ----

◊
 N=                 8             9             10            11
----------------------------------------------------------------------

V 1(N)             20.2222       22.2222       24.2222       26.2222
V 2(N)             10.2222       12.2222       14.2222       16.2222

D 1(N)             2             2             2             2
D 2(N)             2             2             2             2

        ----                  ----                  ----


DO YOU WISH ANOTHER OPTION FOR THE SAME MATRICES?(#Y# OR #N#)
?"N"
DO YOU WISH CALCULATIONS FOR OTHER VALUES? (#Y#,#N#)
?"N"

◊
 1225 HALT
>SYS

!BYE
09/15/'71 15:51
```

APPENDIX PART I

RELIABILITY PREDICTION STUDIES OF COMPLEX SYSTEMS HAVING MANY FAILED STATES

John deMercado

Abstract

In this paper, the theory of discrete Markov Processes is used to develop methods for predicting the reliability and moments of the first time to failure of complex systems having many failed states. It is assumed that these complex systems operate in a repair environment and are composed of subsystems that have known constant failure and repair rates.

Specifically, complex systems composed of any finite number of subsystems are considered. The complex system at any time, can be in any one of $r(r \geq 1)$ acceptable states or in any of $m(m \geq 1)$ failed states. The methods presented for the reliability modelling of such complex systems, assume a state behaviour that is characterizable by a stationary Markov process (also called Markov chain) with finite-dimensional state space and a discrete time set.

It is shown that once the matrix of the constant failure and repair rates of the subsystems is known, and the state assignment is made, then it is a straightforward matter to obtain the probabilistic description of the complex system.

The author is with the Canadian Government, Department of Communications, Ottawa, Ontario, Canada. He is also adjunct professor in the Faculty of Engineering of Carleton University.
Paper submitted January 25th, 1971. Manuscript revised July 15th, 1971.

Introduction

It has been shown [1], [2] that the reliability modelling of complex systems that operate in a repair environment, and whose subsystems have known constant failure and repair rates, can be accomplished via a linear matrix calculus and use of elements of the theory of stationary Markov processes. The methods that exist for modelling such complex systems may be summarized as follows: Let the complex system have r <u>acceptable states</u> A_i (i = 1, ..., r), which form the set A, and let all failed states be lumped into a single <u>failed state</u> F. Then methods exist for obtaining a time dependent reliability function R(n), defined as the probability that the complex system is in some acceptable state in A at time n. Methods also exist, which allow computation of the moments of the first time to failure, that is, the moments of the first time that the complex system passes from acceptable states in A to the single lumped failed state F. These methods all suffer from a number of obvious limitations, first of all the lumping of failed states into a single failed state conceals the relative importance of the different types of failure modes that are present in any complex system. Secondly, no techniques are provided for computing the important moments of the first time the complex system passes from specified acceptable states to specified failed states.

In this paper, the above approach is extended to include complex systems having m(m 1) failed states in the set F. Methods are presented for obtaining a time dependent reliability function for such complex systems, as well as the moments of the first time to (a particular) failed state, as well as the moments of the first time to failure (any state).

The construction of a model for predicting the behaviour of such a complex system poses three distinct problems. The first two are in effect specification problems. The first of these is the <u>state assignment</u> problem, that is the enumeration of the states that suffice to characterize the various operating modes of the complex system. The method for making such a state assignment will depend on the specification of the structure and operation of the given complex system. The second problem involves the determination of meaningful numerical estimates of the one step state transition probabilities[1]. This is the, so called <u>general inference</u>[2] problem |3, pp 69-70|[3] for Markov processes. The third problem which is the one this paper addresses, involves the application of techniques from the theory of stationary Markov process to develop methods for obtaining apriori, state probability functions, a reliability function, and estimates of the moments of the first time that it takes the complex

---

1) The one step state transition probabilities are constant dimensionless and are obtained by multiplying the constant failure or repair rates, (whichever are appropriate) by the "unit of time" (for example, I hour, I day, etc.)

2) Howard |14| in Chapter 6, gives a special and different definition of <u>inference</u>. His book also contains a wealth of Markov models having immediate applications in reliability theory.

3) Numbers in brackets | |, refer to the references.

system to pass from one state to another. We have assumed that the solution to the state assignment problem as well as the general inference problem is known. That is there exists a state characterization of the complex system and the matrix |M| of one step state transition probabilities.

In section I, the basic definitions of the elements of the Markov model of a complex system are presented. It is shown that the matrix[4)] |M| of one step state transition probabilities, that is of the failure and repair rates of the subsystems can be partitioned into four matrices |A| , |B| , |O| , |I| . In later sections it is shown that such partitioning is sufficient to use all the methods presented herein.

In section 2, it is pointed out that the state probability functions $\overline{s}(n)$ are obtained simply by taking the n^{th} power of the matrix |M| . Then once the set A of acceptable states is known, the time dependent reliability function R(n) is shown to be the sum of these state probability functions over the set A. Thus the reliability function R(n), is the probability that at time n, the complex system is operating acceptably.

In example (I), at the end of the paper, it is obvious that another possible interpretation of R(n), in the context of a telecommunication network, is to interpret R(n) as the probability that two points i and j within the telecommunication network will remain connected for time n.

---

4) Capital letters in square brackets denote matrices; the bar on top of a letter denotes a vector. Certain results presented in this paper were also given in an earlier report |9|.

In section 3, the steady state transition failure probabilities p_{ij} are derived. p_{ij} is the probability that the complex system, will eventually pass from acceptable state A_i to failed state F_j. A theorem is presented which shows that the $(r \times m)$ matrix $|P|$, of these steady state failure probabilities is obtainable directly in terms of the matrices $|B|$, $|I|$ and $|A|$, which are the partitions of $|M|$. The concept of an evolution diagram, as introduced by Girault |6|, is utilized to prove this theorem. These evolution diagrams provide a useful conceptual aid for establishing many interesting results in the theory of stationary Markov processes.

In section 4, the steady state transition probability failure functions $p_{ij}(n)$ are derived. $p_{ij}(n)$ is the probability that the complex system will pass after n units of time from acceptable state A_i to failed state F_j. A theorem is presented which shows that the $(r \times m)$ matrix $|P(n)|$ of these transition probability failure functions is expressible directly in terms of the matrices $|A|$ and $|B|$.

In section 5, a method is presented for obtaining the (pseudo) generating functions $g_{ij}(z)$ that give the time moments $\tau_{ij}(k)$ of the random variables $\tau_{ij} \equiv$ "first time from acceptable state A_i to failed state F_j". It is shown that these moments are obtained in the usual manner, that is, by differentiating the (pseudo) generating functions. A theorem is presented, which shows that the $(r \times m)$ matrix $|G(z)|$ of these generating functions is a simple linear function of the matrices $|A|$, $|B|$ and $|I|$.

In section 6 the exit probability functions $w_i(n)$ are derived. $w_i(n)$ is the probability that the complex system will pass from the successful state A_i, into any failed state in F in time n. A theorem is presented which shows that the $(r \times 1)$ column vector $\overline{W}(n)$ of the exit probability fucntions is a simple function of the matrices $|A|$ and $|B|$.

In section 7, a method is presented for computing the generating functions $c_i(z)$ that give the moments $\tau_i(k)$ of the random variables τ_i ≡ "first exit time from acceptable state A_i into the class F". A theorem is presented which relates the generating function $g_{ij}(z)$ of section (5) to the generating function $c_i(z)$. Another theorem is presented which shows that the $(r \times 1)$ vector $\bar{C}(z)$ of these generating functions is a simple linear function of the matrices $|A|$, $|B|$ and $|I|$.

A computer program |15| has been developed for computing all the results presented in this paper. The numerical results at the end of the paper were obtained using this program. The program is written in <u>Basic</u> and has been compiled on a sigma 7 computer.

I. Preliminaries

In developing the reliability model we use a stationary Markov process $S(\cdot)$, defined on a discrete finite dimensional state space $A \cup F$, and a discrete time set T. The random variable $S(n)$ is called "state of complex system at time n". We will derive for each state $A_j \in A$ and $F_j \in F$, state probability functions $s_i(n)$, $s_j(n)$, defined as[5)]

$$s_i(n) \equiv \text{Prob} \quad \{S(n) = A_i\} \quad , \quad n \in T \qquad \text{---------- } 1$$

$$s_j(n) \equiv \text{Prob} \quad \{S(n) = F_j\} \quad , \quad n \in T \qquad \text{---------- } 2$$

It is well known |6,8|, that if the set of states in A, form a transient class (ie, are acceptable states), and if the states in F are absorbing states (ie, are failed states), then the one step transition probabilities between the states $A \to A$, $A \to F$, $F \to F$ and $F \to A$ can be defined as follows.

$A \to A$

The one step state transition probabilities between states A_i, A_k of A denoted by a_{ik}, are the elements of a (rxr) matrix $|A|$ and are defined as

$$a_{ik} \equiv \text{Prob} \quad \{S(n+1) = A_k \mid S(n) = A_i\} \quad , \quad \begin{array}{l} i=1,\text{--},r. \\ k=1,\text{--},r. \end{array} \qquad \text{------ } 3$$

---

5) The subscript i, (i=1,--,r) refers to states in A (i.e., acceptable states) and the subscript j, (j=1,--, m) refers to states in F (the failed states).

$\underline{A \rightarrow F}$

The one step state transition probabilities from transient states $A_i \in A$ to absorbing (failed) states $F_j \in F$, denoted by b_{ij} are the elements of a (rxm) matrix $|B|$, and are defined as

$$b_{ij} \equiv \text{Prob} \quad \{S(n+1) = F_j \mid S(n) = A_i\}, \qquad \begin{matrix} i=1,---,r \\ j=1,---,m \end{matrix} \qquad ----- \ 4$$

$\underline{F \rightarrow F}$

The one step state transition probabilities between absorbing (failed) states F_j, F_u of F denoted by δ_{ju}, are the entries of a (mxm) unit matrix $|I|$, and are defined as

$$\text{Prob} \quad \{S(n+1) = F_u \mid S(n) = F_j\} = \delta_{ju} \begin{cases} = 1, j=u \\ = o, u \neq j \end{cases} \qquad ----- \ 5$$

$\underline{F \rightarrow A}$

Since transitions from failed states in F to transient (acceptable) states in A are not permitted,[6] the one step state transition probabilities from $F_j \in F$ to $A_i \in A$ are all zero. That is they form a (mxr) null matrix $|0|$, because

$$\text{Prob} \ \{S(n+1) = A_i \mid S(n) = F_j\} = 0, \ \forall \ F_j \in F, \ A_i \in A \qquad ------- \ 6$$

6) The methods presented in this paper could be further generalized by allowing transitions among the failed states of F. That is by replacing the matrix $|I|$ by some general matrix. Howard |14|, presents in Chapters 5 and 6, a lucid presentation of the situation when there is a general matrix representing transitions between failure states, i.e., degrees of progressive failure (absorption).

Thus, we have that the one step transition matrix $|M|$ for the Markov processes $S(\cdot)$ with state space $A \cup F$, can be partitioned into four matrices $|A|$, $|B|$, $|I|$, $|O|$ as

$$|M| \equiv \begin{array}{r} A \to A \{ \\ \\ F \to A \{ \end{array} \left| \begin{array}{c:c} |A| & |B| \\ \hdashline |O| & |I| \end{array} \right| \begin{array}{l} \} A \to F \\ \\ \} F \to F \end{array} \qquad \text{---------} \; 7$$

The following definitions make it possible to interpret (3) through (7) in the context of the reliability model of a complex system having A_i, $i=1,---r$ acceptable states, and m failed states, F_j, $j= 1, ---, m$.

<u>Definition 1</u> <u>Acceptable State</u> The <u>transient</u> state $A_i \varepsilon A$ is called an <u>acceptable state</u>, if it characterizes some acceptable working mode of the complex system.

<u>Definition 2</u> <u>Failed State</u> The <u>absorbing state</u> $F_j \varepsilon F$ is called a <u>failed state</u>, if it characterizes some unsatisfactory mode of operation of the complex system.

---

2. <u>State Probability & Reliability Functions</u>

Let $\overline{s(n)}$, be the $(1\times(r+m))$ vector of state probabilities defined by (1) and (2). Then it is well known |6, Page 56| that

$$\overline{s}(n) = \overline{s(o)} \quad |M|^n \quad \text{----------------} \; 8$$

where $\overline{s}(o)$ is the vector of the <u>initial</u> (time n=o) <u>state probabilities</u>. We can now immediately define a reliability function R(n) for the complex system as

<u>Definition 3</u> <u>Reliability Function R(n)</u> The reliability function R(n) is the probability that time n the complex system is operating acceptably, that is, is in some acceptable state, thus

$$R(n) = \text{Prob } \{S(n) \in A\} \text{ ------------ } 9$$

alternatively then

$$R(n) = \sum_{i=1}^{r} s_i(n) \text{ ------------------- } 10$$

Thus in order to obtain R(n), it is necessary only to raise the matrix $|M|$ to the n^{th} power, multiply by $\overline{s}(o)$ and then sum the elements of the set $\{s_i(n),\ i=1,---,r\}$. There are several well known methods |11,12,13| yielding closed form expressions for $|M|^n$ and therefore for $\overline{s}(n)$ and R(n)

---

3. <u>The Steady State Transition Failure Probabilities</u>

In this section, a method is presented for computing the steady transition probability p_{ij}, $(i=1,--,r;j=1,--,m)$ that a complex system that starts in acceptable state A_i will eventually end up in a specified failed state F_j. In what follows, it will be shown that once the partition of $|M|$ has been carried out as shown in (7), it is a simple computational matter to obtain these probabilities. Formally, defining P_{ij} as

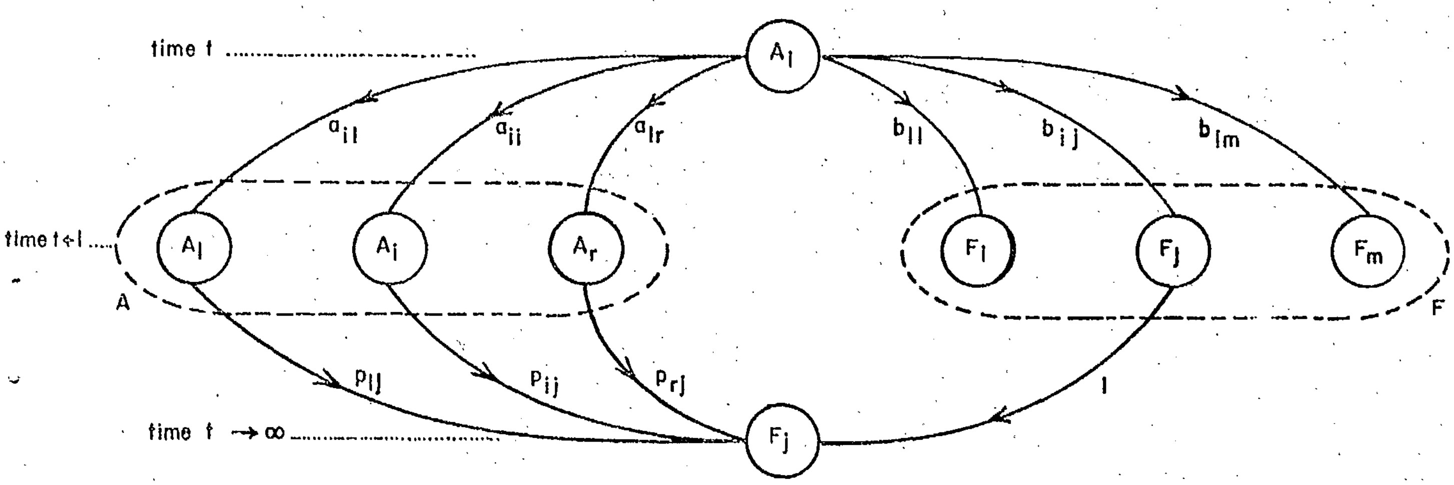
$$p_{ij} \equiv \text{Prob } \{S(\infty) = F_j \mid S(n) = A_i\} \text{------------ } 11$$

and denoting the (r x m) matrix $|p_{ij}|$ by $|P|$, we have,

<u>Theorem I</u> For a complex system with r acceptable states and m failed states operating in a repair environment, and having subsystems with known constant failure and repair rates (that is, with known matrix $|M|$), the $(r \times m)$ matrix $|P|$ satisfies

$$|P| = \Big|\, |I| - |A| \,\Big|^{-1} |B| \qquad \text{-------------- } 12$$

<u>Proof</u> The proof of this and other theorems in this paper is facilitated by formally introducing <u>evolution diagrams</u> |6 pp 74-78|. Consider then Evolution Diagram I, which shows the eventual possible evolutions from state $A_i \varepsilon A$ to state $F_j \varepsilon F$.



<u>EVOLUTION DIAGRAM I</u>

From the above diagram, summing the transmittances of the paths incident on node F_j from A_i, gives

$$p_{ij} = \sum_{k=1}^{r} a_{ik}\, p_{kj} + b_{ij}, \qquad (j=1,----, m) \text{------ } 13$$

obviously such a diagram can be constructed for every $A_i \varepsilon A$ and every $F_j \varepsilon F$ and therefore then (13) can be written in matrix form as

$$|P| = |A|\,|P| + |B|$$

or $\text{---------------- } 14$

$$\Big|\,|I| - |A|\,\Big|\,|P| = |B|$$

which completes the proof QED.

---

4. The Transition Probability Failure Functions

In this section, a method is presented for computing the transition probability failure functions $p_{ij}(n)$, $i=1,---,r$; $j=1,---,m$. Specifically, $p_{ij}(n)$ is the probability that at time n, the complex system is in failed state $F_j \varepsilon F$ given that at time n=0, it was in acceptable state $A_i \varepsilon A$.

Formally

$$p_{ij}(n) = \text{Prob}\,\{ S(t+n) = F_j \mid S(t) = A_i \}, \quad \begin{matrix} j=1,---,m \\ i=1,---,r \end{matrix} \text{ ---- } 15$$

and denoting the $(r \times m)$ matrix $|p_{ij}(n)|$ by $|P(n)|$, we have

<u>Theorem 2</u> For a complex system with r acceptable states and m failed states, and having subsystems with known constant failure and repair rates (that is with known matrix $|M|$), the $(r \times m)$ matrix $|P(n)|$ satisfies

$$|P(n)| = |A|^{n-1}|B| + |P(n-1)| \qquad \text{------------- } 16$$

<u>Proof</u>

Compare (4) and (15), it is immediately apparent

$$|P(1)| \equiv |B|, \; |P(o)| \equiv |0| \qquad \text{----------------- } 17$$

Then from (7)

$$|M|^n \equiv \left| \begin{array}{c:c} |A| & |P(n-1)| \\ \hdashline |0| & |I| \end{array} \right|$$

taking the n^{th} power of $|M|$, using (17), we find

$$|P(n)| = \Big| |I| + |A| + |A|^2 + -- + |A|^{n-1} \Big| |B| \qquad \text{----- } 18$$

which can also be written as (16)

QED

<u>Comments</u>

(a) $p_{ij}(n)$, is the probability that the complex system will pass from acceptable state A_i to failed state F_j in n units of time. Thus letting τ_{ij} be the (pseudo) random variable "time taken to go from state A_i to state F_j", we have that $p_{ij}(n)$ is the probability distribution function of τ_{ij}; that is

$$p_{ij}(n) = \text{Prob}\ \{\ \tau_{ij} = n\ \} \qquad \text{-----------------} \qquad 19$$

(b) Since $|P| = \lim\limits_{n \to \infty} |P(n)|$, from (18) we find

$$|P| = \sum_{n=0}^{\infty} |A|^n\ |B|\ ; \qquad |A|^o \equiv |I|$$

this is an infinite geometric series whose sum is

$$|P| = \big|\ |I| - |A|\ \big|^{-1} |B|$$

which is (12) as obtained previously

5. Moments of the First Time to Failed State

In this section expressions are derived for the (pseudo) generating functions $g_{ij}(z)$ for the moments $\tau_{ij}(k)$, k=1, ,--n, of the (pseudo) random variables τ_{ij}, These random variables are defined as, $\tau_{ij} \equiv$ "first time from acceptable state A_i to failed state F_j".

These moments are the moments of the first time the complex system passes from state $A_i \varepsilon A$ to failed state $F_j \varepsilon F$.

Since the discrete time approach is being used, it is standard practice to define the generating function $g_{ij}(z)$ for these moments in terms of its one sided z-transform. That is, the generating function $g_{ij}(z)$ is defined as

$$g_{ij}(z) = \sum_{n=1}^{\infty} z^n \, p_{ij}(n) \qquad \text{------------------ 20}$$

Definition 4. Moments of First Time to Failed State

The moments $\tau_{ij}(k)$, k=1,...n, of the first time to failed state, are defined as the moments of the first time the complex system passes from acceptable state $A_i \varepsilon A$ to failed state $F_j \varepsilon F$. These moments are obtained from the generating function (20) in the conventional way

$$\tau_{ij}(k) = \frac{d^k}{dz^k} (g_{ij}(z)) \Big|_{z=1} , \quad k = 1,2,\ldots n \qquad \text{------- 21}$$

The following theorem shows how to obtain the (pseudo) generating functions $g_{ij}(z)$ in terms of matrices $|A|$ and $|B|$, without the need for evaluating infinite series of the form (20).

Let $|G(z)|$ and $|\zeta(k)|$ be the $(r \times m)$ matrices of respectively the generating fucntions, and moments of time to first specific failure, then

Theorem 3 The moments $|\tau(k)|$ satisfy

$$\tau(k) = \frac{d^k}{dz^k} |G(z)| \Bigg|_{z=1}, \quad k=1, 2, ---, n.$$

where

$$|G(z)| = \frac{z}{1-z} \Big| |I| - z|A| \Big|^{-1} |B| \quad \text{---------------- } 22$$

Proof .

expanding (20) and using (12), we have

$$|G(z)| = \sum_{n=1}^{\infty} z^n |A|^{n-1} |B| + \sum_{n=1}^{\infty} z^n |P(n-1)|$$

in the above, the second term |14, pg. 45| is $z|G(z)|$ and the first is $z \Big| |I| - z|A| \Big|^{-1} |B|$ QED

6. The Exit Probability Functions

In this section equations are derived for the exit probability functions $w_i(n)$, defined as

$$w_i(n) = \text{Prob}\{S(n+t) \in F \mid S(t) = A_i\} \quad \text{---------- } 23$$

$$= \sum_{j=1}^{m} \text{Prob}\{S(n+t) = F_j \mid S(t) = A_i\} \quad \text{--- } 24$$

Thus, $w_i(n)$, is the probability that the complex system will pass from acceptable state A_i, into the set F in n units of time. Obviously comparing (15) and (24), we have

$$w_i(n) = \sum_{j=1}^{m} p_{ij}(n) \qquad \text{------------------} \; 25$$

Note (25), states that $w_i(n)$ is the sum of the probabilities $p_{ij}(n)$ on the set F.

Letting $\overline{W}(n)$ be the (r x 1) column vector of the exit probability functions, we have

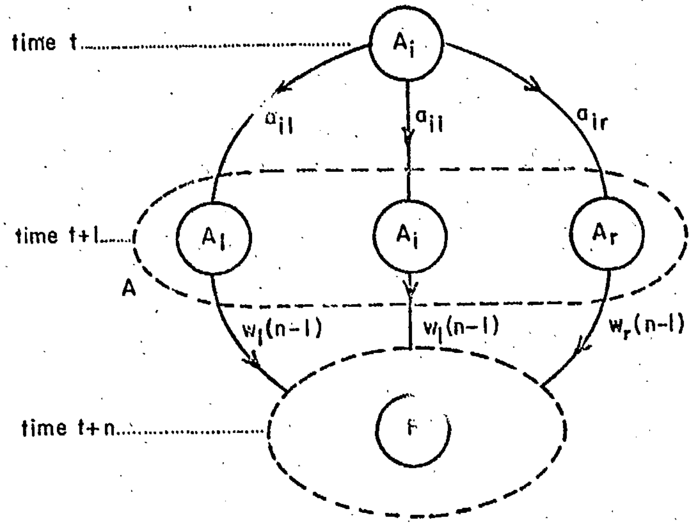
<u>Theorem 4</u> The exit probability functions (23), satisfy

$$w_i(1) = \sum_{j=1}^{m} b_{ij}, \qquad i=1,---,r \qquad \text{-------} \; 26$$

$$\overline{W}(n) = |A| \; \overline{W}\,(n-1) \qquad \text{-----------------} \; 27$$

<u>Proof</u>

Result (26) follows by (4) and (24) after letting n=1, in (25). To prove (27) consider Evolution Diagram 2.



EVOLUTION DIAGRAM 2

The above diagram enumerates the possible evolutions in n steps from any state $A_i \varepsilon A$ to the class F of failed states. Then, summing the transmittances of the paths incident on F, gives for each of the r states $A_i \varepsilon A$, an expression for $w_i(n)$, namely

$$w_i(n) = \sum_{k=1}^{m} a_{ik} \; w_k(n-1), \quad i=1,---,r$$

QED.

7. Moments of the First Exit Time From Acceptable Class A

In this section equations are derived for the generating functions $c_i(z)$ for the moments $\tau_i(k)$, k=1,---,n of the random variables τ_i ≡ "first exit time from acceptable state A_i into failed class F".

Obviously $w_i(n)$ is the probability distribution function of this random variable τ_i, that is

$$w_i(n) = \text{Prob}\{\tau_i = n\}$$

The moments, $\tau_i(k)$, k=1,---,n are the moments of the first time the complex system passes from state $A_i \varepsilon A$ into the class of failed states F. In the reliability literature, these moments are called moments of the first time to failure [8].

Since the discrete time approach is being used, we again define the generating function $c_i(z)$ in terms of its one sided z transform. The generating function $c_i(z)$ is therefore

$$c_i(z) = \sum_{n=1}^{\infty} z^n w_i(n) \qquad \text{------------------ } 28$$

Definition 5. Moments of the First Time to Failure

The moments of the first time to failure, are defined as the moments of the first time the complex system passes from acceptable state A_i to any failed state in F. These moments are obtained from the generating function (28) as

8) In particular, the first moment, is the mean time to first failure.

$$\tau_i(k) = \frac{d^k}{dz^k}(c_i(z))\Big|_{z=1} \qquad k=1,2,\ldots n \quad i=1,\ldots.r$$

The following theorem establishes the relationship between the generating functions $g_{ij}(z)$ and $c_i(z)$ or, equivalently, the relationship between $\tau_{ij}(k)$ and $\tau_i(k)$.

Theorem 5 The generating functions $g_{ij}(z)$ and $c_i(z)$ are related as

$$c_i(z) = \sum_{j=1}^{m} g_{ij}(z) \quad \text{------------------- } 29$$

or equivalently[9]

$$\tau_i(k) = \sum_{j=1}^{m} \tau_{ij}(k) \quad \text{------------------- } 30$$

Proof

Substituting (25) into (28) gives

$$\tau_i(z) = \sum_{n=1}^{\infty} \sum_{j=1}^{m} z^n p_{ij}(n) \quad \text{---------- } 31$$

Interchanging the order of summation in (31)

$$c_i(z) = \sum_{j=1}^{m} \sum_{n=1}^{\infty} z^n p_{ij}(n) \quad \text{------------- } 32$$

substituting (20) into (32) we obtain (29), and (30) follows, by definition.

QED.

9) Obviously using 22 we can immediately evaluate 30.

Comment

Therefore (29) can be computed directly once (21) has been evaluated, or directly, in terms of the matrices $|A|$ and $|B|$, as given in theorem 6 below.

Letting $\overline{C}(z)$ be the $(r \times 1)$ column vector of the generating functions $c_i(z)$, we have

Theorem 6 Let $\overline{\tau}(k)$ be the $(r \times 1)$ vector of the moments $\tau_i(k)$. For a complex system operating in a repair environment and having r acceptable and m failed states and known matrix $|M|$, these moments are

$$\overline{\tau}(k) = \frac{d^k}{dz^k}(\overline{C(z)})\Big|_{z=1}$$

with

$$\overline{C}(z) = \frac{z}{1-z)}\Big|\,|I| - z\;|A|\,\Big|^{-1} \overline{B}' \qquad \text{----------- } 33$$

where

$$\overline{B}' = |b_1', \ldots\ldots, b_r'|^{\dagger}$$

and

$$b_i' = \sum_{j=1}^{m} b_{ij}, \qquad i=1,---,r$$

Proof (33) follows directly from (22) using 24. QED.

Example 1

Consider the following portion of a telecommunication network (Figure 1). The state assignment that describes the operational aspects that we are interested in for this network is shown in Figure 2.

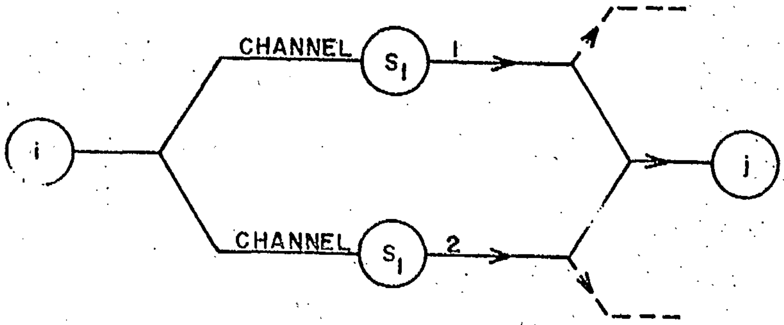


FIGURE 1

State Assignment

State	Word Description
A_1	Both S_1 and S_2 provide a path from i to j
A_2	S_1 fails and the only path is provided by S_2. Repairs to S_1 are not yet started.
A_3	Repairs to S_1 start. S_2 is still providing the connections between i and j.
F_1	S_2 fails before repairs to S_1 have begun.
F_2	S_2 fails before repairs to S_1 are completed.

Figure 2

The transition graph, drawn from Figure 2 is shown below in Figure 3.

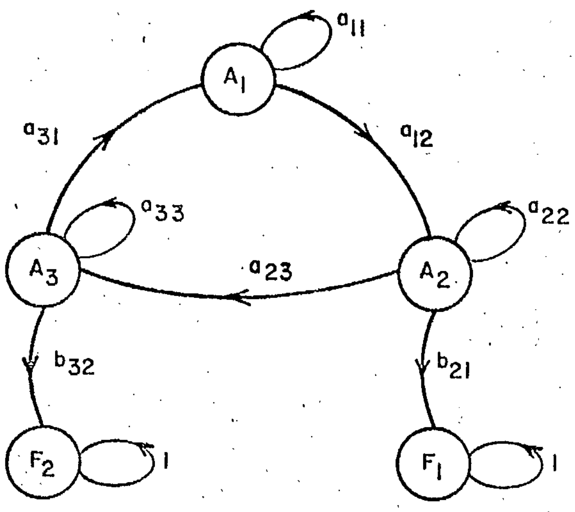


FIGURE 3

For computational purposes we take the failure, repair and delay-repair rates, λ, μ, ρ to be λ = .002/hr. μ = .004/hour ρ = .2/hr. The $|M|$ matrix for this example is therefore obtained as shown in Figure 4 .

$|M| \equiv$

	A_1	A_2	A_3	F_1	F_2
A_1	.998	.002	0	0	0
A_2	0	.798	.2	.002	0
A_3	.004	0	.994	0	.002
F_1	0	0	0	1	0
F_2	0	0	0	0	1

Figure 4

A computer simulation |15|, for $|P(10)|$, $|P(50)|$, $|P|$ as well as the reliability function R(n) for three different initial state vectors $\overline{s}(o)$, is shown in Figure 5.

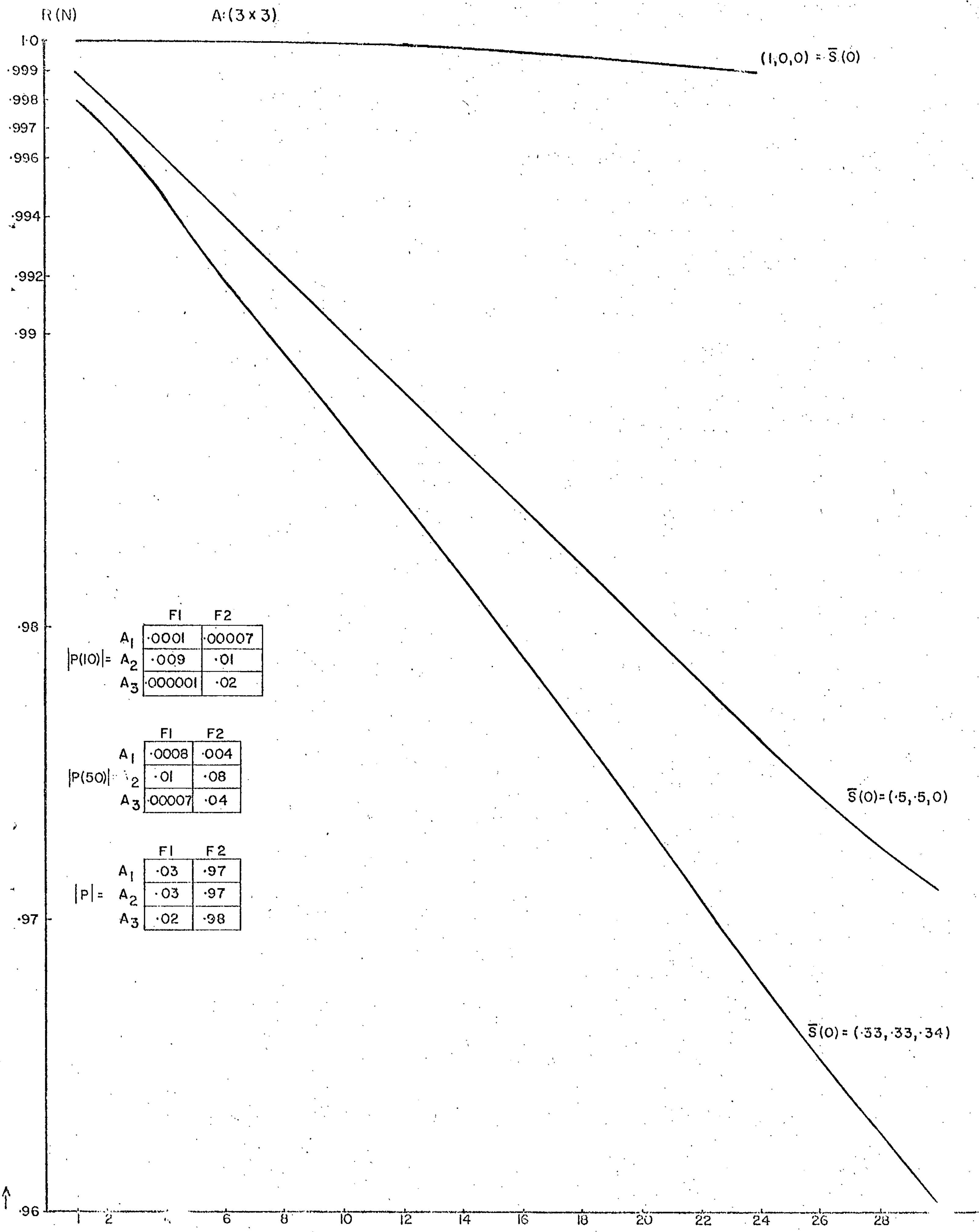

R(N)
A:(3 x 3)
(1,0,0) = $\overline{S}(0)$
$\overline{S}(0)$=(·5,·5,0)
$\overline{S}(0)$=(·33,·33,·34)
1·0
·999
·998
·997
·996
·994
·992
·99
·98
·97
·96
N=
1 2 6 8 10 12 14 16 18 20 22 24 26 28
F1 F2
|P(10)|= A1 ·0001 ·00007
A2 ·009 ·01
A3 ·000001 ·02
F1 F2
|P(50)| A1 ·0008 ·004
2 ·01 ·08
A3 ·00007 ·04
F1 F2
|P|= A1 ·03 ·97
A2 ·03 ·97
A3 ·02 ·98

FIGURE 5

Example 2

A more general system reliability problem yielded the transition graph shown in Figure 6. The corresponding matrices $|A|$ and $|B|$ are shown in Figure 7.

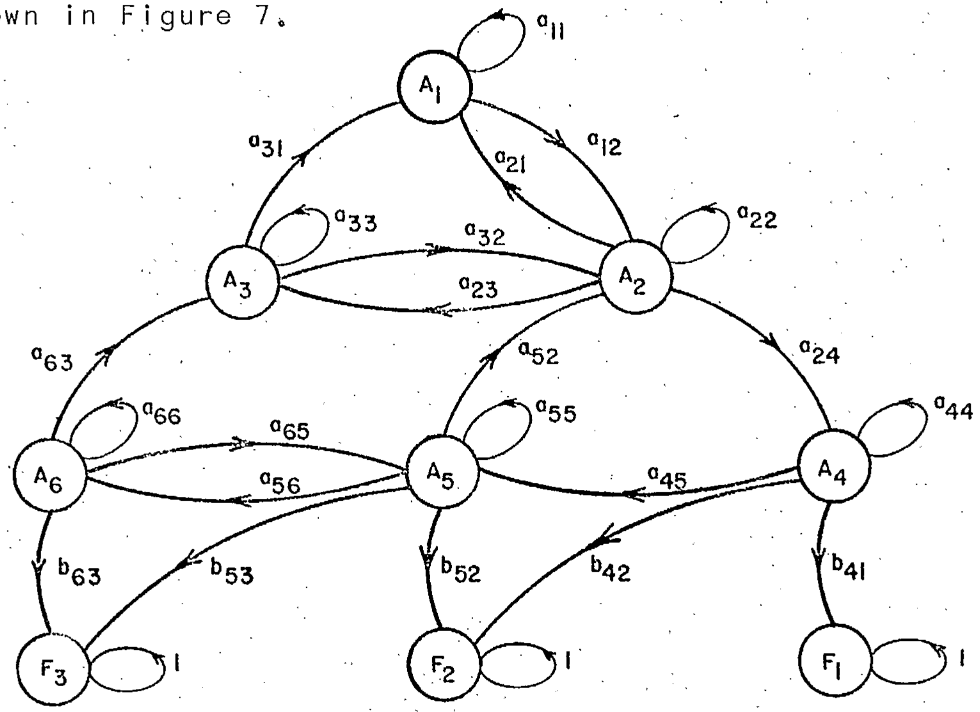


FIGURE 6

$|M| =$

	A_1	A_2	A_3	A_4	A_5	F_1	F_2	F_3
A_1	.90	.045	0	.055	0	0	0	0
A_2	.2	.45	.15	.1	0	.1	0	0
A_3	.6	1	.3	0	0	0	0	0
A_4	0	0	0	.55	.2	0	.15	0
A_5	0	.4	0	0	.2	.1	0	.1
A_6	0	0	.45	0	.1	.25	0	.2
A_7	0	0	0	0	0	0	1	0
A_8	0	0	0	0	0	0	0	0
A_9	0	0	0	0	0	0	0	1

Figure 7

Computer silulation [15] for this matrix $|M|$, yielded the following graph, (Figure 8), of the transition probability failure functions $|P(n)|$. Figure 9 shows the matrices $|P(20)|$ and $|P|$ as well as the Reliability Function R(n) for three different initial state vectors $\overline{s}(o)$.

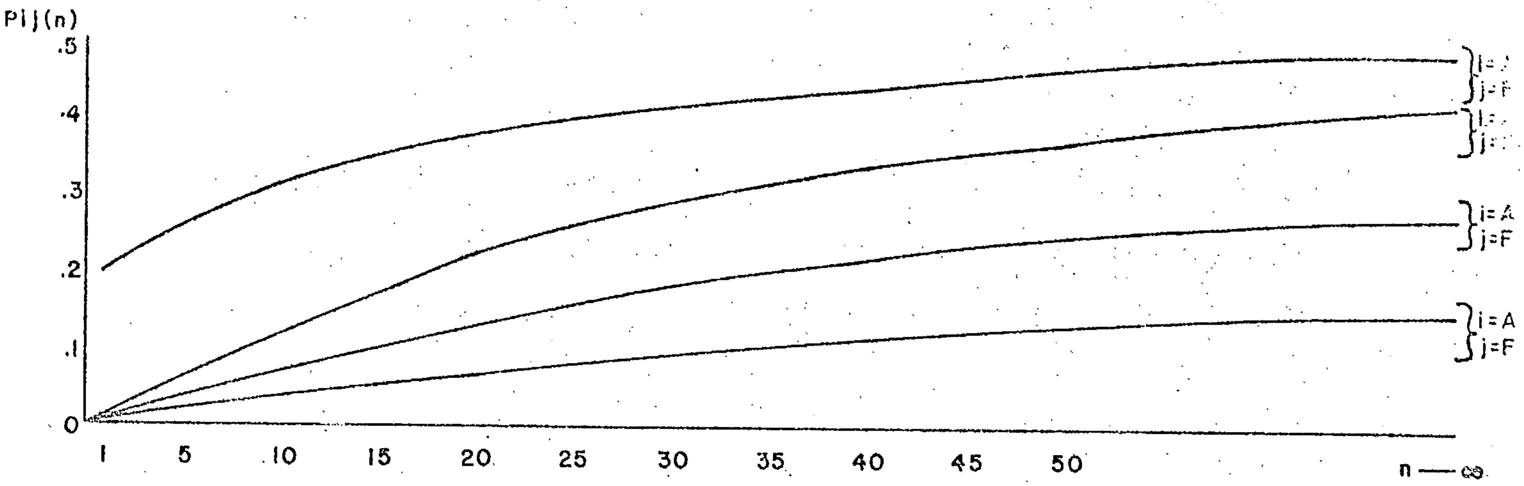


FIGURE 8

Acknowledgements

The author has benefited greatly in many discussions on the modelling of large scale systems with Professor George Glinski and Dr. Douglas Parkhill. He wishes to thank Mr. R. Robert for doing the numerical computations in this paper.

A:(6×6)

$|P(20)| =$

	F1	F2	F3
A_1	·23	·23	·078
A_2	·22	·23	·13
A_3	·22	·22	·08
A_4	·38	·39	·099
A_5	·12	·38	·23
A_6	·14	·17	·34

$|P| =$

	F1	F2	F3
A_1	·42	·43	·15
A_2	·40	·41	·19
A_3	·42	·43	·15
A_4	·44	·44	·12
A_5	·23	·50	·27
A_6	·28	·32	·40

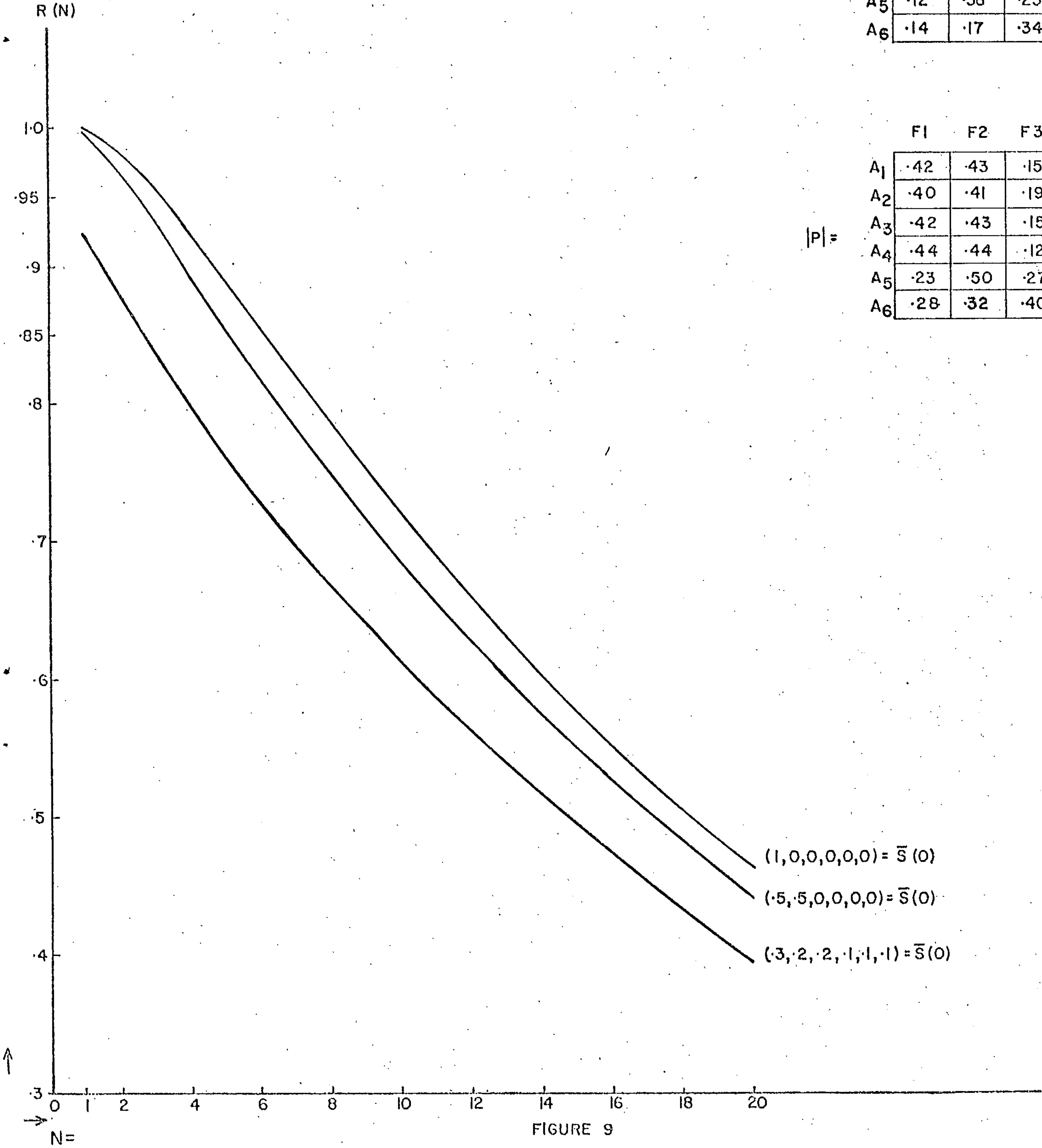


FIGURE 9

References

|1| J. deMercado — Contributions To Reliability Prediction Theory Ph.D. Thesis, Department of Electrical Engineering, University of Ottawa, Ottawa 2, Canada.

|2| L. Thin Htun — Reliability Prediction Techniques for Complex Systems. IEEE Transaction on Reliability theory. August 1966.

|3| B.R. Bhatt — Some Properties of Regular Markov Chains. Annals of Mathematical Statistics, pp. 59-70, 1960.

|4| K. Lai Chung — Markov Chains with Stationary Transition Probabilities, Springer-Verlag, 1967.

|5| G. Glinski, J. deMercado, P.M. Thompson — A Diffusion Method for Reliability Prediction. IEEE Transactions on Reliability Theory, November 1969.

|6| M. Girault — Stochastic Processes, Springer-Verlag, 1966.

|7| W. Feller — Introduction to Probability Theory, Vol. II, John Wiley, 1967.

|8| J. Kemeney, J. Snell, H. Mirkil, G.L. Thompson — Finite Mathematical Structures, Prentice Hall, 1963.

|9| G.S. Glinski, J. deMercado — Reliability Prediction Techniques For Systems with many failed states, University of Ottawa, Electrical Engineering Department, TR No. 70-1, Jan. 1970. Ottawa 2, Ontario, Canada-

|10| M. Pease, Jr. — Methods of Matrix Algebra, Academic Press 1965

|11| R. Bellman — Introduction to Matrix Analysis, McGraw Hill 1960

|12| N.V. Faddeeva — Computational Methods of Linear Algebra. Dover 1959.

|13| L.A. Pipes — Matrix Methods for Engineering, Prentice Hall, 1963.

|14| R.A. Howard — Dynamic Probabilistic Systems, John Wiley, 1971. (Vol. I)

|15| J. deMercado, R. Robert — Reliability Modelling & Simulation Package, Department of Communications, Summer 1971. (available from author).

|16| P. Derusso, R. Roy, C. Close — State Variables for Engineers, John Wiley, 1967.

SECTION 11

PROGRAM NAMES:

SHORT 1(,K)

SHORT 2(,K)

DESCRIPTION

SHORT1(,K) AND SHORT2(,K)

PROGRAM DOCUMENTATION & DESCRIPTION

These programs synthesize communications networks given the communications centers, the terminal capacity requirements and the cost constraints. The requirements do not vary with time and shortest path methods are used to construct the required networks.

The program SHORT1(,K) synthesizes a network in which all the requirements are to be met at the same time while SHORT2(,K) constructs a time-shared network in which only two terminals communicate with one-and-other at one time. Since the input-output sections of these programs are identical (except for the program descriptions) they will be described together. The names SHORTx(,K) and SHORTxB will refer to both programs where x = 1 or 2.

The main feature of these packages is that by using the Batch Time-Sharing Monitor (B.T.M.) of the Xerox Sigma-7, the programs can be used in conversational mode. Once the user initiates the programs the packages will offer certain output options and will ask for input data as the program sequence proceeds. Terminal requirement data and cost constraint data may either be entered conversationally from the keyboard or automatically from a prepared data file. This second option will save the user time in reloading large data files.

LOGGING ON:

For "dial-up terminals" one of the following telephone ports should be dialed:

9-828-2754	(low speed)
996-7051, EXT: 505, 506, 507 or 508	(high speed) (day)
996-6723; 996-6724; 996-6725	(high speed) (night)

Once the monitor responds with "!LOGIN: ", the user should type: "PLANNING,1004S, POLICY " to get logged on. Whenever the system responds with an "!", then it indicates that we are in "Executive Mode", that is, the highest level of system control.

COMPILATION:

The source file, SHORTx is on disk as is the binary version SHORTxB. If for some reason SHORTxB is lost or accidentally altered, it can be re-created using the following monitor commands (note: underlined symbols are those that the system supplies automatically and the "↵" indicates that a carriage return is required).

```
!ASSIGN M:  SI,(FILE,SHORTx),(PASS,K)*↵
!ASSIGN M:  BO,(FILE,SHORTxB)↵
!FORTRAN
 OPTIONS:  NOLS,BO↵

 ** END OF COMPILATION **
 ** END OF COMPILATION **
 ** END OF COMPILATION **
 ** END OF COMPILATION **
[** END OF COMPILATION **] **
```

* Although the password K is typed by the user, it will not appear on the printout at the terminal.

** "END OF COMPILATION" appearing five times means successful compilation for SHORT2 while it is required only four times for SHORT1.

LOADING:

If the binary file is satisfactory, the user may load the program and begin execution. If all information is to be entered from the terminal, use procedure 'A'; and if the data is to be entered from a disk file named DATA, say, then use procedure 'B'.

PROCEDURE 'A' -

!LOAD

ELEMENT FILES: SHORTxB

OPTIONS:

F: 1

F:

SEVERITY LEVEL = 0

XEQ? Y

PROCEDURE 'B' -

!LOAD

ELEMENT FILES: SHORTxB

OPTIONS

F: 1

F:

SEVERITY LEVEL = 0

XEQ? Y

At this point, the program begins execution. The user is referred to the examples for compilation, loading and execution to aid him in using the program.

PROGRAM INPUT:

When the program asks for logical options, the user should respond by typing either "YES" or "NO". The examples show the responses to all these options.

1- The matrix size is the first data input requested. The user is to input an integer not greater than 15. This value is then stored internally in the variable N and is used to determine the size of the input matrices below. N is also the number of nodes in the network.

USING A SEPARATE DATA FILE:

The user may wish to enter data for the terminal capacity and arc cost matrices from the file "DATA". He may build this file using the system editor and make alterations to these values later if he wishes. The user is referred to the B.T.M. users' manual for use of the editor.

If we have an N node network, N lines of N decimal values per line should be entered for the terminal capacity matrix and in a similar fashion, N^2 integer values should be entered for the arc cost matrix. Each value on a line should be separated from the following one by a comma.

The user is reminded that carriage returns must be removed from the ends of lines for data files to be processed properly by the system.

LOGGING OFF:

When the monitor has returned with a "!", the user should respond with "BY" if he wishes to "get off" the system.

EXAMPLE: PROGRAM SHORT1(,K)

```
ASSIGN M:SI,(FILE,SHORT1),(PASS,)

!ASSIGN M:BO,(FILE,SHORT1B)

!FORTRAN
OPTIONS: NOLS,BO
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **

!LOAD
ELEMENT FILES: SHORT1B
OPTIONS:
F:1
F:

SEV.LEV. = 0
XEQ? Y
```

```
THIS NETWORK SYNTHESIS PACKAGE IS AT YOUR COMMAND!
PLEASE INPUT DATA AS REQUESTED BY THE PROGRAM!
DO YOU WISH TO SEE THE PROGRAM DESCRIPTION?
ANSWER YES OR NO
?YES


THIS PROGRAM SYNTHESIZES A COMMUNICATIONS NETWORK
GIVEN THE COMMUNICATIONS CENTRES, THE TERMINAL CHANNEL
CAPACITY REQUIREMENTS AND THE ARC  COST CONSTRAINTS.
THE REQUIREMENTS ARE MET SIMULTANEOUSLY AND THEY DO
NOT VARY WITH TIME. SHORTEST PATH TECHNIQUES ARE USED
TO ARRIVE AT THE SOLUTION.

INPUT:
     N-(INTEGER)-THE NUMBER OF COMMUNICATIONS CENTRES:
      N IS THE DIMENSIONALLITY OF THE MATRICES BELOW.
     T-(DECIMAL)-THE TERMINAL CAPACITY MATRIX:
      EACH ENTRY, T(I,J), CONTAINS THE VALUE OF
      REQUIRED CHANNEL CAPACITY FROM TERMINAL I TO TERM-
      INAL J.
     K-(INTEGER)-THE ARC COST MATRIX:
      EACH ENTRY, K(I,J), IS THE COST PER UNIT CAPACITY
      ON ARC (I,J).

OUTPUT:
     R-(DECIMAL)-THE REQUIRED CAPACITY MATRIX:
      EACH ENTRY,    R(I,J), IS THE CHANNEL CAPACITY
      OF ARC (I,J) THAT IS REQUIRED FOR THE SOLUTION.
     TT-(DECIMAL)-TOTAL NETWORK COST:
      TT=SUM(R(I,J)*K(I,J))  FOR ALL I AND J.


ARE THE REQUIREMENTS AND THE COSTS SYMETRICAL?
ANSWER YES OR NO
?NO


INPUT THE NUMBER OF NODES PLEASE!
?3
```

```
PLEASE INPUT  9 FLOATING POINT VALUES
3 PER LINE TO FILL THE TERMINAL CAPACITY MATRIX!


 1:
?0,2,5
 2:
?10,0,2
 3:
?8,6,0


PLEASE INPUT  9 INTEGER VALUES
3 PER LINE TO FILL THE ARC COST MATRIX!


 1:
?0,2,4
 2:
?1,0,1
 3:
?2,5,0


DO YOU WISH TO REVIEW YOUR INPUT?
ANSWER YES OR NO
?YES


THE TERMINAL CAPACITY MATRIX:

   .00  2.00  5.00
 10.00   .00  2.00
  8.00  6.00   .00

THE ARC COST MATRIX:

   0     2     4
   1     0     1
   2     5     0


DO YOU WISH TO RE-ENTER YOUR DATA?
ANSWER YES OR NO
?NO


THE REQUIRED CAPACITY MATRIX BELOW REPRESENTS
THE NETWORK THAT SIMULTANEOUSLY SATISFIES THE REQUIREMENTS!

    .0     13.0       .0
  10.0       .0      7.0
  14.0       .0       .0


TOTAL NETWORK COST IS-      71.00


DO YOU WISH TO RESTART THIS PROGRAM?
ANSWER YES OR NO
?NO
 *STOP* 0
```

EXAMPLE: PROGRAM SHORT2(,K)

```
!ASSIGN M:SI,(FILE,SHORT2),(PASS,)

!ASSIGN M:BO,(FILE,SHORT2B)

!FORTRAN
OPTIONS: NOLS,BO
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **

!LOAD
ELEMENT FILES: SHORT2B
OPTIONS:
F:1
F:

SEV.LEV. = 0
XEQ? Y


THIS NETWORK SYNTHESIS PACKAGE IS AT YOUR COMMAND!
PLEASE INPUT DATA AS REQUESTED BY THE PROGRAM!
DO YOU WISH TO SEE THE PROGRAM DESCRIPTION?
ANSWER YES OR NO
?YES


THIS PROGRAM SYNTHESIZES A COMMUNICATIONS NETWORK
GIVEN THE COMMUNICATIONS CENTRES, THE TERMINAL CHANNEL
CAPACITY REQUIREMENTS AND THE  ARC COST CONSTRAINTS.
THE REQUIREMENTS DO NOT  VARY WITH TIME AND THEY ARE
TIME-SHARED IN SUCH A WAY THAT ONLY TWO TERMINALS MAY
COMMUNICATE WITH ONE AND OTHER AT ONE TIME. SHORT-
EST PATH TECHNIQUES ARE USED TO ARRIVE AT THE SOLUTION.

INPUT:
      N-(INTEGER)-THE NUMBER OF COMMUNICATIONS CENTRES:
       N IS THE DIMENSIONALITY  OF THE MATRICES BELOW.
      T-(DECIMAL)-THE TERMINAL CAPACITY MATRIX:
        EACH ENTRY,     T(I,J), CONTAINS THE VALUE OF
       REQUIRED CHANNEL CAPACITY FROM TERMINAL I TO TERM-
       INAL J.
      K-(INTEGER)-THE ARC COST MATRIX:
       EACH ENTRY,K(I,J), IS THE COST PER UNIT CAPACITY
      ON ARC (I,J).

OUTPUT:
      R-(DECIMAL)-THE REQUIRED CAPACITY MATRIX:
       EACH ENTRY,   R(I,J), IS THE CHANNEL CAPACITY
       OF  ARC(I,J) THAT IS REQUIRED FOR THE SOLUTION.
      TT-(DECIMAL)-THE TOTAL NETWORK COST:
       TT=SUM(R(I,J)*K(I,J)) FOR ALL I AND J.
```

```
 INPUT THE NUMBER OF NODES PLEASE!
?3


 PLEASE INPUT  9 FLOATING POINT VALUES
 3 PER LINE TO FILL THE TERMINAL CAPACITY MATRIX!


 1:
?0,2,4
 2:
?7,0,1
 3:
?6,5,0


 PLEASE INPUT  9 INTEGER VALUES
 3 PER LINE TO FILL THE ARC COST MATRIX!


 1:
?0,4,2
 2:
?1,0,3
 3:
?1←2,1,0



 DO YOU WISH TO REVIEW YOUR INPUT?
 ANSWER YES OR NO
?YES


 THE TERMINAL CAPACITY MATRIX:

    .00  2.00  4.00
   7.00   .00  1.00
   6.00  5.00   .00


 THE ARC COST MATRIX:

    0     4     2
    1     0     3
    2     1     0


 DO YOU WISH TO RE-ENTER YOUR DATA?
 ANSWER YES OR NO
?NO
```

```
THE REQUIRED CAPACITY MATRIX BELOW SATISFIES THE
TIME-SHARED REQUIREMENTS AND REPRESENTS THE DESIRED NETWORK!

      .0        .0       4.0
     7.0        .0        .0
      .0       6.0        .0


TOTAL NETWORK COST IS-      21.00


DO YOU WISH TO RESTART THIS PROGRAM?
ANSWER YES OR NO
?NO
*STOP* 0

!BYE
09/13/'71 16:32
RAD SPACE 8 RELEASED
CPU TIME 3.202
I/O WAIT TIME 0.801
MON SERVICES 1.111
!
```

SECTION 111

PROGRAM NAME:

NETSYM 1(,K)

DESCRIPTION

NETSYM1(,K)

PROGRAM DOCUMENTATION & DESCRIPTION

This program synthesizes a communications network given the communications centers, the terminal capacity requirements and the cost constraints. The requirements do not vary with time and they are time-shared in such a way that only two terminals may communicate with one-and-other at one time. The method is dependent on the presence of redundant terminal requirements.

The main feature of this package is that by using the Batch Time-Sharing Monitor (B.T.M.) of the Xerox Sigma-7, the program can be used in conversational mode of the "question-answer" type. Once the user initiates the program, the package will offer certain output and will ask for input as the program sequence proceeds. Terminal requirement and cost constraint data may either be entered conversationally from the keyboard or automatically from a prepared data file. The second option will save the user time in reloading large data files.

LOGGING ON:

For dial-up terminals one of the following telephone ports should be dialed:

9-828-2754	(low speed)
996-7051, Ext., 505, 506, 507 or 508	(high speed) (Day)
996-6723-5	(high speed) (night)

Once the monitor responds with "!" LOGIN:, the user should type in: PLANNING,1004S,POLICY to get logged on.

COMPILATION:

The source file, NETSYM1 is on disk as is the binary version NETSYMB1. If for some reason, NETSYMB1 is lost or accidnetally altered, it can be re-created using the following monitor commands.

NOTE: Underlined symbols are those that the system automatically supplies.

```
! ASSIGN M:SI,(FILE,NETSYM1),(PASS,K*)
! ASSIGN M:BO,(FILE,NETSYMB1)
! FORTRAN
 OPTIONS: NOLS,BO

** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
```

* IMPORTANT The "K" in the first line is the password to NETSYM 1. Although it is typed by the user, it will not appear on the terminal.
If "END OF COMPILATION" appears five times, then, this phase is successful.

LOADING:

If the binary file is satisfactory, the user may load the program and begin execution. If all information is to be inputted from the terminal, use procedure 'A' or if the terminal requirement data and the cost data are on a disk file called DATA, say, then use procedure 'B'.

PROCEDURE 'A' -

```
!LOAD
ELEMENT FILES:   NETSYMB1
OPTIONS:
F: 1
F:
SEVERITY LEVEL = 0
XEQ? Y
```

PROCEDURE 'B' -

```
!LOAD
ELEMENT FILES:  NETSYMB1
OPTIONS:
F: 1 =  DATA, IN
F:
SEVERITY LEVEL = 0
XEQ?  Y
```

At this point, the program begins execution. The user is referred to the appendix for examples of compilation, loading and execution to aid him in using the program.

PROGRAM INPUT:

When the program asks for logical options the user should respond by typing either "YES" or "NO". The example shows the response to all these options.

DATA INPUT:

1- The matrix size is the first data input requested. The user is to input an integer not greater than 15. This value is then stored internally in the variable N and is used to determine the size of the input matrices that follow. N is also the number of nodes in the network.

2- The program then requests the user to input values to fill the terminal capacity matrix, T which is an N by N matrix. Each entry, T (I,J) represents the requirements between node I and node J that the synthesized network must satisfy. By entering N lines (each one terminated by a carriage return) with floating point values per line (each value separated from the next by a comma) the user fills T row by row.

If a floating point number happens to be a whole number, a decimal point need not be entered.

Due to limitations of page width, the values may range from 999.9 to 000.0 (ie - one decimal fraction of precision)

3- The final data request asks the user to fill the arc constraint matrix, K which is again a N by N matrix. Each entry, K (I,J) may represent the cost per unit capacity of arc (i,j). Integer values are entered as for T above and the range of K is from 0 to 9999.

PROGRAM OUTPUT:

The main output of the program is the matrix R which is the "Required Capacity Matrix". The value of each entry, R (I,J) represents the capacity that must be built from I to J. If all the entries of R are constructed, a network will result that satisfies the requirements with the given constraints.

An optional output is the "Calculated Terminal Capacity Matrix". This matrix is calculated from the solution matrix and is a check on the synthesized network.

USING A SEPARATE DATA FILE:

The user may wish to enter data for the terminal capacity matrix and the arc cost matrix from the file "DATA". He may build this file using the system editor and make alterations to these values later if he wishes. The user is referred to the B.T.M. users' manual regarding the use of the editor.

If we have a N node network,N lines of N decimal values per line should be entered for the terminal capacity matrix and similarily N^2 integer values should be entered for the arc cost matrix. Each value on a line should be separated from the following one by a comma.

The reader is reminded that carriage returns must be removed from the ends of the lines of data files. "*CR OFF" in "! EDIT " will accomplish this.

LOGGING OFF:

When the monitor has returned with a " ! " (this can always be obtained by typing "escape - escape") the user should respond with "BY" if he doesn't wish to continue processing.

EXAMPLE: PROGRAM NETSYM1(,K)

```
!LOGIN: ←PLANNING,1004S,POLICY
ID= 3
!ASSIGN M:SI,(FILE,NETSYM1),(PASS,)

!ASSIGN M:BO,(FILE,NETSYMB1)

!FORTRAN
OPTIONS: NOLS,BO
*** END OF COMPILATION ***
*** END OF COMPILATION ***
*** END OF COMPILATION ***
*** END OF COMPILATION ***
*** END OF COMPILATION ***

!LOAD
ELEMENT FILES: NETSYMB1
OPTIONS:
F:1
F:

SEV.LEV. = 0
XEQ? Y


 THIS NETWORK SYNTHESIS PACKAGE IS AT YOUR COMMAND!
 PLEASE INPUT DATA AS REQUESTED BY THE PROGRAM!

 DO YOU WISH TO SEE THE PROGRAM DESCRIPTION?
 ANSWER YES OR NO
?YES


 THIS PROGRAM SYNTHESIZES A COMMUNICATIONS
 NETWORK GIVEN THE COMMUNICATIONS CENTERS, THE TERMINAL
 CHANNEL CAPACITY REQUIREMENTS AND THE ARC  CONSTRAINTS.
 THE REQUIREMENTS DO NOT VARY WITH TIME AND THEY ARE
 TIME-SHARED IN SUCH A WAY THAT ONLY TWO TERMINALS MAY
 COMMUNICATE WITH EACH OTHER AT ONE TIME. THE METHOD
 IS DEPENDENT ON THE PRESENCE OF REDUNDANT TERMINAL
 REQUIREMENTS. FURTHER PROGRAM DESCRIPTION IS AVAILABLE
 IN THE PROGRAM DOCUMENTATION.

 INPUT:
         N-(INTEGER)-THE NUMBER OF COMMUNICATIONS CENTERS:
           N IS THE DIMENSIONALITY OF THE MATRICES BELOW.
         T-(DECIMAL)-THE TERMINAL CAPACITY MATRIX:
           EACH ENTRY,     T(I,J), REPRESENTS THE REQUIRED
           CHANNEL CAPACITY FROM TERMINAL(CENTER) I TO
           TERMINAL J.
         K-(INTEGER)-THE ARC CONSTRAINT MATRIX:
          EACH ENTRY, K(I,J), REPRESNTS THE RELATIVE VALUE
          OF CONSTRUCTING THE ARC (I,J). LOW VALUES OF K(I,J)
          GIVE THOSE ARCS HIGH CONSTRUCTION PRIORITIES.

 OUTPUT:
         R-(DECIMAL)-THE REQUIRED CAPACITY MATRIX:
            EACH ENTRY,    R(I,J), REPRESENTS THE CHANNEL CAPACITY
            THAT MUST BE CONSTRUCTED FROM I TO J IN ORDER TO
            ACHIEVE THE DESIRED SOLUTION
```

```
 INPUT THE NUMBER OF NODES PLEASE!
?4


 PLEASE INPUT  16 FLOATING POINT VALUES
  4 PER LINE TO FILL TERMINAL CAPACITY MATRIX


  1:
?0,2,2,2
  2:
?3,0,4,6
  3:
?3,7,0,8
  4:
?3,5,4,0


 PLEASE INPUT  16 INTEGER VALUES
  4 PER LINE TO FILL ARC CONSTRAINT MATRIX


  1:
?0,2,1,2
  2:
?1,0,2,0
  3:
?2,0,0,0
  4:
?2,0,1,0


 DO YOU WISH TO REVIEW YOUR INPUT?
 ANSWER YES OR NO
?YES


 THE TERMINAL CAPACITY MATRIX:

    .00  2.00  2.00  2.00
   3.00   .00  4.00  6.00
   3.00  7.00   .00  8.00
   3.00  5.00  4.00   .00


 THE ARC CONSTRAINT MATRIX:

     0      2      1      2
     1      0      2      0
     2      0      0      0
     2      0      1      0


 DO YOU WISH TO RE-ENTER YOUR DATA?
 ANSWER YES OR NO
?NO
```

```
THE CAPACITY MATRIX BELOW REPRESENTS THE NETWORK
THAT EXACTLY SATISFIES THE TERMINAL REQUIREMENTS!


     .0     .0    2.0     .0
    3.0     .0     .0    4.0
     .0    4.0     .0    4.0
     .0    3.0    2.0     .0

DO YOU WISH TO SEE THE CALCULATED TERMINAL
CAPACITY MATRIX? ANSWER YES OR NO.
?YES


RESULTANT TERMINAL CAPACITY MATRIX:

     .0    2.0    2.0    2.0
    3.0     .0    4.0    6.0
    3.0    7.0     .0    8.0
    3.0    5.0    4.0     .0


DO YOU WISH TO RESTART THIS PROGRAM?
ANSWER YES OR NO
?NO
 *STOP* 0

!BYE
09/14/'71 09:02
```

SECTION 1V

PROGRAM NAME:

NETPLAN(,K)

DESCRIPTION

DOCUMENTATION

NETWORK FLOW AND EXPANSION ROUTINES

WRITTEN UNDER CONTRACT

TO THE

DEPARTMENT OF COMMUNICATIONS

GOVERNMENT OF CANADA

FACULTY OF MANAGEMENT SCIENCES

UNIVERSITY OF OTTAWA

JULY 1971

This document supersedes all previous documentation issued for this study by the faculty.

Table of Contents | Page

Abstract

The flow network optimization package NETPLAN has been designed for use in a time-sharing environment. The package accepts as input a description of a single-commodity network with upper and lower bounds on capacity in each arc. In response to user requests it provides the following facilities:

(a) Checking of the described network for feasibility.

(b) Adjusment of flows in individual arcs to minimize total transport cost, with total circulation from source to sink to be either specified or maximized.

(c) Adjustment of capacities of individual arcs to either maximize total throughput achieved for a given incremental investment, or else to expand the given network to achieve a given total throughput with minimum investment.

Both transport and investment costs in each arc are considered to be linear functions of arc flow.

The program is written in Extended FORTRAN IV H for the Sigma 7 system.

Introduction

The planner of a communication network is faced with problems of a degree of complexity insurmountable in the present state of the art. Nevertheless, network planning is done, albeit in fragmentary and heuristic fashion. At this moment, then, it is not a question of a complete answer to the total problem, but rather of any improvement in the tools available to the planner which might yield significant returns.

The planning package presented as NETPLAN is a first step in that direction. By means of this package it is now possible for the planner to evaluate the performance of the network as a whole, at least in relation to traffic flowing between a particular pair of nodes. Within the limitations of the model, multi-source network synthesis and analysis is possible as well.

The state of the art will allow analysis and optimal synthesis of multi-source, multi-terminal networks, but this was beyond the scope of the present contract. Past this are the non-linear and dynamic aspects of network planning which were discussed in the preamble to the Faculty's proposal of last February.

Despite all of these limitations, the authors do feel that in NETPLAN they offer a significant improvement over prior, more piecemeal techniques. The package we offer is flexible in application. Used parametrically, it can bring the Department NETPLAN as an useful first addition to the Department's planning tools.

General Description of the Package

The flow of logic in NETPLAN is depicted in general outline in Flow Chart A. The action begins in the main program with the requesting of input from the user. Most of this input can then be reviewed prior to further action; this is useful where input has come from file and/or where the printing of input requests has been suppressed.

The main program sets up an artificial return arc from sink to source, in order to express the problem, for the moment in circulation form. It is at this stage, if optimization of the existing network is to be carried out, that the choice of priority between flow maximization as opposed to cost minimization is carried out.

Whatever program options have been specified, the program then calls upon subroutine PRIME to establish feasibility. Prime first builds tables in scratch memory that speed up the scanning of arcs adjacent to a given node. Then flows, as initially described by the user in his input, are checked to see if they are conservative (i.e. if inflow equals outflow at every node, including flow along the return arc).

If flows are not conservative, the program discards N-1 of them, where N is the number of nodes. The arc flows to be discarded are determined by constructing a tree spanning the network and rooted at the source node. The N-1 flows are then solved for in terms of the remaining ones.

The user may ask whether there is any point in supplying initial flows if some are to be discarded. The usefulness of this feature will be realized, however, when a series of problems is being run, each consisting of minor modifications to the same basic network. In this case, it will be possible to start from the solution to a previous problem, and save computer time by so doing.

The conservative set of arc flows, once obtained, forms the initial input to the algorithm which generates the first feasible solution to the network. This algorithm, contained in subroutine NETFLO as called from PRIME, will be described in more detail below; it and the relevant theorems come from Ford and Fulkerson(1), pages 50-53.

Whether or not the feasible solution exists upon return to the main program, the user has the option of reviewing the output of the feasible flow-generation algorithm.

If the user has requested optimization of the existing network, this is now accomplished by a call from the main program, again to NETFLO. The algorithm for this optimization is also taken from Ford and Fulkerson, this time from pages 162-169. Upon return from NETFLO, reporting of arc flows, is automatic.

If the user has requested expansion of an existing network, the first step taken is to delete the artificial return arc, returning the problem to a source-sink formulation. This occurs after optimization of the existing network if this was requested; otherwise, it comes immediately after the call to PRIME. Next, the limit the desired budget or throughput is read in.

The program calls upon BUDGET to compute the expansion. The algorithm upon which BUDGET is based comes from Fulkerson's paper (2). Certain modifications have been included in this algorithm, both in the interests of computational efficiency and to take account of possible lower bounds on arc flows. Reporting of the computed expansion is done directly from BUDGET.

BUDGET, like NETFLO, accepts any feasible set of initial flows. This set may have come from NETFLO via PRIME if flow optimization was not requested, or else may be the optimal set of flows produced by NETFLO the second time it was called.

Upon return from BUDGET, the user is offered the chance to re-run the program. He can re-run either with the same options or with new ones. If input was from file rather than keyboard, it will remain so; the file will be rewound so that data is read starting again at the beginning of the file.

If the user wishes, therefore, to run with new data in a file, he is best advised to escape to the monitor system, SAVE current contents of core, EDIT his data file, RESTORE the program, and proceed. A more useful aspect of the re-run facility as set up is that it will provide for reading of data previously stored on disk by a modified version of NETPLAN. If BUDGET did the storing, for instance, the revised package would be useful for multi-terminal network synthesis.

The general logical flow of NETPLAN is depicted in Flow Chart A.

The Algorithms: Preliminary Discussion

This section will summarize briefly the contents of the several papers upon which NETPLAN is based.

It will do so in conceptual fashion - the equations can safely be left to the references already cited. We begin with the description of the type of network which NETPLAN will deal with, and point out first some features of feasible flows.

A network in the first instance is defined by a set of nodes, and by a set of arcs linking these nodes. In our case we are interested in directed networks. In such networks, one node of each arc is its initial node, while the other is the terminal node; there is a specific forward direction along the arc.

In such a network, it is always possible to pick out paths: that is, sequences of arcs, with each successive pair of the sequence linked by a common node. There is no requirement that in passing from one end of the path to the other all the arcs be traversed in the same sense; some may be forward arcs of the path, some reverse arcs.

Let us now consider the idea of flows on the arcs. Departing somewhat from the usage of Ford and Fulkerson, let us say that a set of flows in a network is any set of numbers, one for each arc.

Let us now suppose that each arc has associated with it two other numbers: a lower limit, and an upper limit greater than or equal to the lower limit. By assuming one more property in our network, we can proceed to define first a conservative, and then a feasible set of flows. The property we require is as follows: all nodes but two of the network are at once both initial nodes to one or more arcs, and terminal nodes to one or more other arcs. One of the two special nodes, called the source, is the initial node to one or more arcs, but is the terminal node to no arc. The other node called the sink, is a terminal node to at least one arc, but never an initial node.

Note a first consequence of this property: any node whatever of the network is connected to any other node by at least one path. Furthermore, any pair of nodes one of which is neither the source or the sink (an internal node) is connected by at least two different paths. This is because each internal node is at the end of at least two different arcs, by assumption. The other ends of these arcs can be linked to any selected node of the network each by at least one path, also by assumption. Completing these paths to the given internal node by means of the two arcs gives two paths from the selected other node. These two paths differ at the very least by the two arcs adjacent to the internal node, hence our assertion is proved.

We now define a concervative set of flows in a network. It is a set of numbers, one for each arc, satisfying the following properties:

(1) For every internal node of the network, the sum of flows entering the node is equal to the sum of flows leaving it. Expressed in our previous language, the sum of flows over the set of arcs terminated by the given node is equal to the sum of flows over the set of arcs initiated by that node.

(11) The sum of flows leaving the source node is equal to the sum of flows entering the sink node.

If there are N nodes in the network, we see that condition (1) amounts algebraically to a set of linear equations in the flows, for each of N-2 internal nodes. Condition (11) becomes one more linear equation, giving a total of N-1 relations in all.

We are now ready to define a <u>feasible</u> set of flows in the network. Such a set has the following characteristics:

(1) It is conservative,

(11) The flow for each arc is at least as large as the lower limit, and no greater than the upper limit.

Condition (11) amounts to two sets of linear inequalities, one for the lower limits, and one for the upper ones. If the limits have been given for some network, the first question that comes to mind is whether any feasible flow exists for that network. Algebraically speaking, this is a question of the consistency of the N-1 equations (1) and the 2M inequalities (11) taken together, where M is the number of arcs - and hence flows - in the network.

In order to consider this question further, let us modify the source-sink network we have already considered. Let us now define a <u>network in circulation form:</u> it is the network obtained by adding one arc to the source-sink network, leading from the sink node back to the source. This <u>return arc</u> will also have associated with it a flow, a lower limit, and an upper limit.

We say a set of flows in a network in circulation form is conservative if the flows in the source-sink network obtained by removing the return arc are conservative. It is easily seen that, where in a source-sink network conservative flows required that total flow leaving the source equal total flow entering the sink, we have two conditions in the circulation-form network.

These are that total flow entering equal total flow leaving for each in turn of the source and the sink node. Thus in a circulation-form network all nodes are internal - a more symmetric situation than the source-sink one.

For the circulation-form network, the definition of a feasible set of flows can be taken as before, with the new meaning of the term "conservative". The conditions for feasibility now amount to N conservation equations - one for each node - and the 2M limit inequalities defined on the arcs. It can be demonstrated that there are in fact at most N-1 independent equations: the conservation equation at the sink node is in effect a summary of what has happened at the source and intervening nodes. This fact has importance when we try to construct feasible flows, as will be described later.

Hoffman (3) has used the theory of linear inequalities to derive the necessary and sufficient condition for the existence of a feasible set of flows in a circulation-form network. This is that if we take any arbitrary subset of nodes and consider the arcs linking that subset with the remaining nodes of the network, then the sum of lower limits on those arcs entering the subset must not exceed the sum of upper limits on the arcs leaving the subset.

It is evident that a network in source - sink form will be feasible that is, at least one feasible set of flows exists, if a circulation-form network derived from it is. On the other hand, given a feasible source-sink network, the feasibility of the circulation-form networks derived from it will depend on the lower and upper limits placed on the return arc.

Derivation of Feasible Flows

Suppose we begin in a network in circulation form with an arbitrary set of flows. Our task is to determine if the network is feasible, and to construct a feasible. Two steps are required. The first is to construct from the original flows a set which is conservative. This, as we shall see, is always possible. Then we build from the second set of flows a third set, satisfying both the conservation conditions and the set of upper and lower limits. This latter task is only possible if Hoffman's condition is met.

As we noted above, the conservation conditions amount to N-1 independent linear equations. Because they are in terms of flows, of which there are M, we see that solution is possible provided M is not less than N-1.

In a circulation-form network, there is a minimum of N arcs (assuming connectivity), so that conservative flows can always be achieved.

How do we solve for these flows? We must arbitrarily set M-N+1 of the flows (that is, leave them at their original values), and solve for the remaining N-1 flows in terms of them. But if we choose the wrong flows to pre-set, we shall find that their elimination from the N-1 equations leaves these equations no longer independent.

How do we choose which flows to solve for then? It can be shown that, if we select any N-1 arcs of the network which form a tree, the flows in these arcs can be found successfully from the conservation equations. Note that a tree of N-1 arcs must necessarily include all N nodes of the network. It is not necessary that all branches of the tree be directed away from the root. Nevertheless, NETPLAN constructs just such a directed tree, in order to simplify subsequent logic. The following algorithm is used:

(a) Mark the source node (the root) as "reached".

(b) Find any node which has been marked "reached" but not "scanned".

(c) Examine all arcs for which the node selected in (b) is the initial node

  (c1): If the terminal node of such an arc is marked, go on to another arc.

  (c2): If the terminal node of such an arc is unmarked, mark it "reached" and include the arc in the tree.

(d) When all arcs leaving the node selected in (b) have been examined, mark that node "scanned".

(e) Terminate when no more nodes can be marked.

If termination occurs before all nodes have been marked, there exists some subset of nodes which cannot be reached from the source. Unless the lower limits on flow in this subset are uniformly zero, or else the necessary loop exists to propagate the required flow, the network will then be infeasible.

In practical problems it is considered likely that such disconnectedness is due to a specification error on the part of the user rather than his intent. Thus, rather than complicate the logic of generating conservative flows, the program rejects the network immediately as incorrectly specified. If such disconnection is not in fact erroneous, the program should be modified to use a non-directed tree to generate conservative flows. On any case, once conservative flows, presence or absence of directed connectivity does not affect the operation of the feasible-flo -generating routine.

Once a tree is defined, solution of the conservation equation is a simple matter of working back from the tips of the branches toward the root. A node at the tip of a branch has associated with it one conservation equation and only one unknown flow. Since all coefficients in the conservation equations are plus or minus unity, solution is just a matter of addition and subtraction.

Having our conservative set of flows, where do we go next? Ford and Fulkerson (1) defined the algorithm which we are about to examine. The essatial principle of this algorithm is to remove violations of lower and upper limits, one by one, while retaining conservation and permitting no new limit violations to occur. If the process is blocked at some point, and constraint violations remain, the network fails to satisfy Hoffman's condition and posesses no feasible set of flows.

The basic working of the algorithm can be seen as follows. Suppose that an arc directed from node A to node B currently has, say, a flow greater than its upper limit, the set of flows being conservative. Suppose we lowered the high flow by some specific amount. Then in order to preserve conservation, the following must happen:

(1) At node A, either the flow in some arc entering A must be reduced, or else the flow in some arc leaving A must be increased.

(11) At node B, either the flow in some arc entering B must be increased, or else the flow in some arc leaving B must be reduced.

Suppose we begin at B and choose some arc other than the original one in which we are trying to reduce the flow.

If the chosen arc is leaving B, as we have just noted, flow will have to be reduced. In order to keep from making the situation worse rather than better, the following choices are necessary:

(i) If flow in the chosen arc is less than or equal to the lower limit already, we must not reduce flow further; we pass on to another arc.

(ii) If flow in the chosen arc is greater than the lower limit, we are free to reduce flow, but not by more than the difference between current flow and lower limit.

Similarly, for an arc entering B, we can increase flow only so long as the upper limit is not exceeded. In actual fact, whatever the arc chosen, the change will not exceed the smaller of the value permitted on that arc, and the value by which the original arc is to be reduced.

Suppose now that we consider a node C, linked to node B by an arc along which a change in flow is possible. Here again, a conservation relation must be preserved. Thus arcs leading to and from C must be examined for the possibility of changes in flow. And if another arc is so chosen, flow in it, too, can be changed by only so much. In fact, as we construct this path of arcs, the maximum possible change in flow will be set by the tightest link- the arc of smallest possible change-along that path.

But where does this process of path-building end? It ends when the path has moved out to the point where the latest link includes node A. For at this point we have a closed loop within the network. By construction, if the final link of that loop enters A, its flow can be reduced; if it leaves A, its flow can be increased. Thus the flow in the original arc can be reduced, and by adjusting flows all along the path we have defined, conservation will be preserved at each affected node in turn. Note that the adjustment is no greater than that permitted by the tightest link in the loop, as before. Thus more paths may be needed before the high flow within the original arc is finally brought within bounds.

The analysis is similar for an arc which originally has a flow less than the lower limit. Thus we have described a procedure which preserves conservation, causes no further violation of any flow limits, and reduces the amount of violation in a particular arc.

But what if we cannot find a closed path such as we have described, such that a non-zero change in flow is possible? Then a subset of nodes of the network can be defined, such that minimum flow into the subset from the rest of the network exceeds the maximum flow out, or vice versa; that is, Hoffman's condition is violated, and no feasible set of flows exists.

The offending subset can be defined as follows:

a) Include the two nodes A and B at the ends of the original offending arc.
b) Include any nodes connected to nodes already in the subset, by arcs along which change in the desired direction is possible.

By hypothesis, the point will eventually reached such that no further nodes can be added to this subset, and the total set of nodes has not been exhausted. Then in order to render the network feasible it will be necessary to increase upper limits or decrease lower limits in arcs connecting the offending subset with the rest of the network.

NETPLAN does not go so far to help the user. If the network proves infeasible, NETPLAN gives details for the original offending arc. The user can then ask for the current status of the whole network to be reported. It is up to the user to construct the offending subset as described above.

Construction of conservative flows in done in subroutine PRIME of NETPLAN. Construction of feasible flows as we have described is done in subroutine NETFLO, provided TASK is set to 1 on entering that routine. Flow Chart B describes the algorithm used, as taken from Ford and Fulkerson (1, pp. 52-53).

DERIVATION of OPTIMAL FLOWS

So far in our discussion, nothing has been said about the cost aspects of networks. One of the purposes of NETPLAN, however, is to solve for the user the problem of maximum flow at minimum cost. More precisely, NETPLAN will find a flow pattern in a circulation-form network which maximizes the flow through the return arc. If alternative patterns with the same maximal flow rate exist, the cheapest one is selected. The cost of a given flow pattern is expressed as the sum over all arcs of unit cost in each arc times the flow in that arc.

The above is a description of what happens internally within NETPLAN. Externally, the user is asked to describe a network in source-sink form. Provided he wishes flow optimization, he is then asked whether he wants to specify a flow rate from source to sink to be achieved at minimum cost; alternatively, he can ask for maximum flow at minimum cost.

Within the program, action is then taken as follows. The network is converted to circulation form by the addition of a return arc. If the user has specified a flow to be achieved, the lower and upper limits of the return arc are set equal to this value, and the unit cost through the return arc is set to zero. If flow is to be maximized, the lower limit on the return arc is set to zero. The upper limit is set to infinity. To provide a financial incentive to maximize flow, the unit cost of flow through the return arc is set to minus infinity.

The program now checks the circulation-form network for feasibility as described above, and creates, if possible, a feasible set of flows. As a point of interest, if flow maximization is not asked for by the user, a circulation-form network is still created in order to check feasibility. In this case the return arc is given the limits appropriate to flow maximization, since these are broad enough not to affect the final judgement of feasibility of the rest of the network. This is the point of the initialization IV = 0 displayed on page A-1, Flow Chart A.

Flow optimization in this program thus begins with a feasible set of flows created previously by the subroutines PRIME and NETFLO within a circulation-form network. Because these flows are feasible, the full generality of the Ford-Fulkerson out-of-kilter algorithm, to the description of which we now turn, is unnecessary. We will therefore describe only that portion of the algorithm which has been implemented in NETPLAN.

To understand the motivation of this algorithm, we suppose that the commodity whose flow we are dealing with can be given a price at each node of the network. If a market really existed at each node for this commodity, it is evident that the following possibilities could hold along any arc leading from node A, say, to some other node B:

(I) If the price of the commodity at B were higher than the price at A plus the unit cost of moving the commodity along the given arc from A to B, there would be an incentive to use that arc to capacity.

(II) If the price of the commodity at B were lower than the price at A plus the unit cost of moving the commodity along the given arc from A to B, there would be an incentive to reduce the flow in that arc to its lower limit.

(III) If there were neither gain nor loss in moving the commodity from A to B along the given arc, then the flow could vary anywhere from lower to upper limit in that arc without affecting total profitability of the network.

One can see that the prices at different nodes would, in a real market, be inter-related by the costs of moving from one node to the other by means of arcs of the network. This, perhaps, will give some intuitive validity to a result which will now be stated. This result comes from the theory of duality in linear programming.

We state the following: for any network for which feasible flows exist, one can find a set of prices and a feasible set of flows, dependent upon each other, such that the profitability of the network is maximized. That is, propositions (I) to (III) above have been taken to their conclusions in deriving the set of flows.

The important point about this statement is that, once prices have been set, profitability of the network will be maximized by operating it at minimum cost. Thus while one pursues an artificially-constructed objective of maximum profitability, one is at the same time achieving the user's original objective.

How, one may ask, does this guarantee maximum flow when the return arc has been so constructed to achieve this? The answer lies in the "cost" of minus infinity given in that case to the return arc. At the point in proceedings where market equilibrium has been reached, in order to achieve this equilibrium on the return arc it will have been necessary to either price the commodity at plus infinity at the sink node, or minus infinity at the source. Whichever node has been so affected will pass incentives to move arc flows to one limit or the other through the network. Indeed, where flow at an upper or lower limit is not possible due to constraints elswhere in the network, the infinite price must be passed to the other end of the arc in question in order to satisfy equilibrium condition (III). Only those arcs which can be taken to the required limit will fail to transmit the infinite price. In such a way does the pricing of the return arc cause the program to define a minimum cut (see (1), P. 11-13), and thence a maximum flow.

With this background, we can now describe the workings of the out-of-kilter algorithm itself. This algorithm is designed to move toward a set of prices and attendant flows which will achieve market equilibrium and thus the user's objectives. It does this by means of series of breakthroughs, where flows are adjusted, and non-breakthroughs, which result in the adjustment of prices.

We begin defining kilter numbers. In an arc where there is a positive market incentive to increased flow (situation (I) above), the kilter number is equal to the product of that unit incentive, times the difference between the upper limit and actual flow in the arc. Where, on the other hand, there is a negative incentive to flow (situation (II) above), the kilter number of the arc is equal to the absolute magnitude of that incentive, times the difference between the actual flow and lower limit for the arc. All other arcs (situation (III)) have zero kilter number.

It can be seen that the kilter numbers are always greater than or equal to zero, and that they provide a rough local measure of market dis-equilibrium. The objective of the program becomes that of reducing all kilter numbers to zero.

The optimization is done in subroutine NETFLO, which was also used to derived feasible flows. As will be seen, the logic of the two tasks is quite similar in many respects. The routine now begins with a set of feasible flows, and a set of node prices (initialized to zero, although this is not necessary to the algorithm).

Any out-of-kilter arc is now located. An out-of-kilter arc is one having a non-zero kilter number. Thus profit-increasing possibilities exist on this arc for the network entrepreneur. If no out-of-kilter arcs exist, the algorithm is finished.

The algorithm now tries to take advantage of the profit-increasing possibilities along this arc. If, for example, the arc corresponds to situation (I) above, the program looks for a way to profitably increase flow through that arc. Such an increase can only come by adjusting flows all along a closed loop of the network in order to preserve conservation, just as was described for the process of finding feasible flows previously.

However, the arcs which may potentially be included in this loop are now subject to economic considerations as well as those of feasibility. Only those arcs are considered, such that a change of flow in the desired direction will result in a (possibly zero) increase in network profitability. Thus flow will be increased only along arcs flowing at less than the upper limit and having zero or positive market incentive to increased flow. Flow will be decreased only along arcs flowing at more than the lower limit and having zero or negative incentive under the current set of prices.

If a loop of eligible arcs is found, the algorithm has achieved what is known as a breakthrough on this case, a flow adjustment along the loop, and in particular in the selected out-of-kilter arc, is possible. The amount of this adjustment is limited to the smallest absolute difference found along the loop between current flow and upper or lower limit as the case may be for a given arc.

Thus if breakthrough occurs, flow in at least one arc in the loop will be pushed against either its upper or lower limit. No flow will be taken outside of one or the other limit, thanks to the selection rule for size of adjustment. Finally, the kilter number in the selected arc and possibly in others will be diminished, since by construction the flow in that arc will have been moved by a non-zero amount in the direction called for by the market incentive.

If the kilter number of the selected arc has been reduced to zero, the program goes on to find another out-of-kilter arc. Otherwise, it proceeds to try to find another loop through which flow in the arc might be adjusted.

The alternative situation to breakthrough is that of non-breakthrough. Here the program has tried to construct a loop of adjustable and profit-maintaining arcs and has failed. On order to make this attempt, the algorithm has, by construction, started at the end of the out-of-kilter arc at which the commodity is most valuable. In situation (I), where arc flow is to be increased, this is the terminal node of the arc. In situation (ii) it is the initial node.

The program has proceeded to test every possible path by which flow could be carried away from the starting node and thence eventually to the other end of the selected arc. In so doing, it has examined all arcs eligible under the rules described above. From nodes reached by these arcs, it has examined other eligible arcs, and so on until it has defined a subset of nodes from which no nodes outside the subset can be reached by eligible arcs. By construction, this subset includes the starting node, but does not include the other end of the selected out-of-kilter arc.

In order to break this impasse, the algorithm adjusts the set of market prices. It does this by examining the arcs linking the "reached" subset of nodes with the rest of the network. It finds that arc which has the minimum market disincentive to moving flow away from the "reached" subset (which was our objective in striving for a breakthrough).

That is, if a given arc leads away from the reached subset and would incur negative profit per unit increase of flow in the direction of that arc, the absolute value of this non-zero profit is considered. If, on the other hand, an arc leads from the rest of the network into the reached subset and would incur positive profit per unit increase of flow in the direction of that arc, then that positive unit profit is considered. The minimum value of all considered incentives and disincentives (greater than zero construction) is selected.

The selected minimum value is now added to the node price at every node not belonging to the reached subset. This does not affect the profitability of flow through arcs, both ends of which are either within or outside the reached subset, since profitability depends upon the difference between node prices and that has not been affected.

Arcs linking the reached subset with the rest of the network have, however, been affected. It is now less unprofitable to move flow away from the reached subset, and less profitable to move it toward that subset. In fact, for at least one of these arcs, the incentive to move flow in either direction has been reduced to zero.

Moreover, all kilter members on the affected arcs have been decreased or left at zero by the price adjustment. Indeed, suppose that we are considering an arc leading out of the reached subset. The arc was not eligible to be included in a loop from one and of the originally-selected out-of-kilter arc. Thus it was either operating at its upper limit with non-negative incentive, or operating at less than upper limit with a negative incentive. An addition to the terminal node price will occur. If the arc was operating at its upper limit, its kilter number remains at zero.

Otherwise, the amount of disincentive is decreased. If the arc was out-of-kilter, this implies a reduction in the kilter number. By the rule for selection of the price increase, the disincentive will at most be reduced to zero.

Similar reasoning applies to an arc leading into the reached node subset. Thus kilter numbers are reduced and at least one is reduced to zero. At least one arc will be opened up to flow from the reached subset outwards. At least one more node will be reached.

If the other end of the selected out-of-kilter arc is added to the reached subset, breakthrough has occured and flows are adjusted accordingly. That or a new out-of-kilter arc is selected as before. Otherwise, the non-breakthrough procedure with its price adjustments is repeated until breakthrough does occur.

Thus the out-of-kilter algorithm constantly works toward increased network profitability until a minimum-cost maximal-flow solution is achieved. Breakthroughs reduce cost through re-adjustment of flows. Non-breakthroughs cause price re-adjustments to move the market toward equilibrium. Flow Chart C gives the algorithm in detail.

LEAST-COST EXPANSION OF AN EXISTING NETWORK

We come now to the description of the workings of the last algorithm used in NETPLAN: that for expansion of an existing network at minimum cost. Again we will take the point of view of an entrepreneur trying to maximize net profits from the delivery and sale of a commodity. But this time the algorithm is more straightforward; it is intuitively more obvious what is being accomplished.

We begin with a network in source-sink form, for which has been given a feasible set of flows. Within NETPLAN, this has been accomplished by creating a return arc, using PRIME and NETFLO as described above to generate feasible flows, and then deleting the return arc. A minimum-cost flow pattern may also have been created previously by a second call to NETFLO, but this is irrelevant given that the flows are feasible.

Besides the flow pattern, we are given the unit cost of expansion of each arc. The cost of added capacity is thus assumed to be a linear function of the incremental capacity. We are therefore beginning with a network in which arcs may or may not be currently used to capacity. For an arc which is not filled, we have in effect available a certain amount of additional throughput at zero cost. Beyond this, added throughput has the unit cost given for the arc.

Let us consider the situations that might face our supposed entrepreneur. One of the following possibilities obtains along any given arc of the network:
(I) Price at terminal node is less than that at initial node. In this case the entrepreneur is losing money on every unit of flow shipped. There will be an incentive for him to reduce flow to the lower limit for the arc.

(II) Price at terminal node equals that at initial node. Here there is no incentive to move flow either way. Any flow between the lower and upper limit for the arc will yield to same zero return.

(III) Price at terminal node exceeds that at initial node, but by less than the cost of expansion. Here there is a positive incentive for the entrepreneur to ship up to the capacity of the arc, since the price difference represents profit to him. It is not worth his while to expand the arc.

(IV) Price at terminal node exceeds that at initial node by the unit cost of expansion. Due to the logic of the algorithm to be used, this is the final case that need be considered. Here the entrepreneur will operate the arc at least to capacity, and does not mind expanding that capacity, since his costs are met.

Once more we appeal to the duality theory of linear programming. Let us stipulate that our entrepreneur has maximized his profits according to some set of market prices. If he has done so while achieving a required throughput from source to sink, he has also minimized costs. If he maximized profits while holding the total cost of expansion within some specified budget, he has also maximized the throughput from source to sink possible with this budget.

We are now in a position to state in general terms how the algorithm works. A detailed exposition will follow. In general, then, the algorithm searches out a succession of sets of paths from source to sink, along each member of which at least one more unit of flow is to be sent. These paths are found by applying the entrepreneur's rules (I) - (IV) described above for maximizing profits. When no further path can be found such that profitability is non-negative for added flows, the market prices at the nodes are adjusted in order to induce further flows in a set of paths, until such time as the throughput or budget objective be met or exceeded. If the objective is over-shot, the full capacity of the latest set of paths found will not be needed, so that interpolation between the latest and next-previous solutions is necessary.

By the nature of the algorithm, unit cost per unit added flow will be monotone non-decreasing in going from one set of paths to the next. The first set chosen will be those paths requiring no additions to arc capacities. Then, when the maximum throughput possible in the original network has been achieved, succeeding sets will include paths along which capacities of some arcs have not yet been used up, while other arcs involve a cost of expansion.

Finally, in what is called an "infinite breakthrough", the cheapest remaining path will be one requiring expansion in every single arc along its whole length. Moreover, all arcs of this path will be forward arcs. It is evident that no matter how much flow is added after an infinite breakthrough has occured, it will be added to arcs of this path alone.

The algorithm begins with all node market prices set to zero. The source node is marked as "reached", but not "scanned", and is also marked as being the first node on the path which will correspond to an infinite breakthrough.

The algorithm now tries to find the remainder of an infinite breakthrough path. Except in the trivial case, none will exist at the outset of the program. Nonetheless, the information stored for nodes reached during this search, at the beginning or at any later stage of the construction, remains valid from then on.

As will be recalled, an infinite breakthrough path consists of forward arcs, all of which are in situation (IV) as described above. That is, the increase in market price from one end to the other of the arc just equals the unit expansion cost along that arc.

The search proceeds by considering all arcs leaving a "reached" but not yet "scanned" node. Any arcs in cost situation (IV) reach new nodes. If these latter are not already marked, they are marked as "reached" and as possibly lying on the infinite breakthrough path. The arcs that reached the respective nodes are noted. The search terminates either when no new nodes have been reached, or else when the sink is reached. In the latter case an infinite breakthrough has occured; final disposition of this case is described later.

If no infinite breakthrough is found, the next step is to search for a finite breakthrough. A finite breakthrough is achieved when a path is found from source to sink along which flow may be increased, and for no arc of which will our hypothetical entrepreneur's profit be decreased. To prepare for this new search, all nodes marked "reached and scanned" in a search for infinite breakthrough revert to "reached only" status. Nodes marked in a previous search for finite breakthrough revert to unmarked status; information about arcs associated with them during that search is discarded.

The search begins with any "reached" but not "scanned" node. If all arcs entering or leaving that node have been examined, the node is marked "scanned". Otherwise some arc is selected. If this arc, in the first case, leaves the selected node, the terminal node is checked. If the terminal node is marked a new arc is taken. Otherwise examination of this arc continues.

The market situation along the arc is now checked. If the price is the same at both initial and terminal nodes (situation (II)), the arc is checked to see if current flow is at the upper limit. If not, the terminal node is marked "reached". The number of the arc being examined is stored for the terminal node. Finally, a maximum permissible flow change and its direction (positive) are stored for the terminal node. This maximum change is the smaller of that already stored for the initial node, or that possible before the current arc is used to its upper limit. This completes processing of the current arc in this case.

If, on the other hand, the current arc is still a forward arc, but the price at the terminal node exceeds that at the initial node, we have either situation (III) or situation (IV). If situation (III) holds, the arc is passed over and another examined. This is because, by the way the algorithm works, a price difference only appears along an arc when it is already against a limit preventing further increase in flow from source to sink. As we have seen in the discussion of situation (III), it is not profitable to raise an upper limit in this case.

We are thus left with the case of a situation (IV) forward arc. Here the arc is already operating at capacity. The cost of any expansion is covered by the market price difference. The terminal node is therefore marked "reached", and the associated arc, direction of flow change (positive), and maximum increase are stored as before, with one difference. By construction, the current arc presents no limit to flow expansion. Thus the maximum permissible change at the terminal node is equal to that already stored for the initial node.

Having discussed all possible cases involving forward arcs, we now consider arcs such that our selected node is terminal rather than initial to them. These arcs would be reverse arcs of a path passing from source node through the selected node to the sink. Thus, to increase flow from source to sink, flow must be decreased in these arcs.

Two operative cases again present themselves; those of situation (II) and of situation (IV). The other two cases again cannot be usefully exploited or do not occur due to the workings of the algorithm. Taking the situation (II) case, we recall that this involves equal market prices at both ends of the arc. The terminal node was the one originally selected.

If, at the opposite end, the initial node is already marked, or if the arc is already flowing at its lower limit, discard the arc. Otherwise, mark the initial node as "reached". Store for it the arc which reached it, the direction of flow change (negative), and a maximum permissible value of the change. This value is the minimum of that already stored for the terminal node, or the difference between current flow and the lower limit in the current arc.

Finally, a situation (IV) reverse arc is, as we recall, one in which the price at the selected terminal node exceeds that at the initial node by exactly the unit cost of expansion for the arc. Moreover, by construction, flow in this arc equals or exceeds the original upper limit.

If the initial node in this case is already marked, or flow in the arc does not exceed the upper limit, the arc is discarded. Otherwise the initial node is marked "reached", and the associated arc, direction of flow change (negative), and maximum permissible flow change are stored. The maximum flow change is the smaller of the value already stored for the terminal node and the difference between the current flow and the original upper limit.

We have now considered all cases which can occur during the search for a finite breakthrough. At some point in this search, either the sink node is reached (finite breakthrough), or else all reached nodes have been scanned and no other nodes can be reached (non-breakthrough). In the case of a finite breakthrough, flows are adjusted along a path from sink to source, by the maximum change amount stored for the sink node, and in the direction and along the arc stored for each succeeding node of the path.

Thus by construction, a finite breakthrough results in the flow in at least one arc being taken to a lower or upper limit, while a non-zero increase in flow from source to sink is achieved. This done, a new search for a finite breakthrough is prepared and executed. A succession of new paths will be generated until a non-breakthrough occurs. All paths generated by this set of breakthroughs will be at least as expensive as those generated in the previous set.

Successive sets of breakthroughs are separated by at least one non-breakthrough, by definition. Fulkerson (2) has shown that, if the required expansion budget or total throughput lies between the corresponding values obtaining at the conclusion of two successive sets of breakthroughs, the required solution is obtained by interpolation between the flow patterns holding at those two times. If, furthermore, an infinite breakthrough has been found before the flow or budget limit has been reached, Fulkerson gives a formula for extrapolation from the last set of flows obtained from a previous finite breakthrough to the objective.

If termination does not save us from the necessity, a non-breakthrough will then require the adjusting of market prices in order to promote a new breakthrough. The adjustment is implemented by considering the set of all nodes which were reached during the previous unsuccessful search for a finite breakthrough. Market prices for nodes within this set are left untouched. For each node outside of the reached set, the market price is adjusted upwards by the same amount.

We turn our attention to the problem of determining the magnitude of this adjustment. In order to do this, we consider that set of arcs which links the reached set of nodes with other nodes of the network. The aim of our adjustment will be to render at least one of these arcs eligible for inclusion in a breakthrough path, as none are now. Furthermore, the adjustment must not be larger than the minimum necessary to achieve this effect. Otherwise cases would arise which have been dismissed above as inoperative.

Let us consider the nature of the arcs of the linking subset, and the effect of the price adjustment upon them. We first consider those arcs leading out of the reached subset of nodes. By the rules of search for breakthrough, none of these arcs can be in situation (IV). Furthermore any of these arcs presently in situation (II) must have flows equal to the original upper limit.

Since the price adjustment will be added at the terminal nodes of these arcs, their situation numbers will either hold constant or increase. Situation (I) arcs will remain in situation (I) or advance to situation (II). If they do advance to situation (II), they will become eligible for a finite breakthrough path, since their flows (as will be shown below) must have been at the lower limits.

Situation (II) arcs will necessarily move to situation (III), where they will be ineligible for consideration, or else possibly to situation (IV). Note that situation (III) arcs so created will have flows at the original upper limits.

Arcs originally in situation (III) will either stay there or move to situation (IV). Whatever the original nature of an arc, it will, correctly, be flowing at its original upper limit when it is changed to a situation (IV) arc. All situation (IV) arcs are eligible for inclusion in a breakthrough path.

Thus, as far as these outward-bound arcs are concerned, our price adjustment quantity must be the smallest quantity such that one or more of the following events will occur:

- a situation (I) arc will move to situation (II);
- a situation (II) arc will move to situation (IV);
- a situation (III) arc will move to situation (IV).

We can now perform a similar analysis of those arcs leading into the reached subset of nodes. In this case, arcs in all four situations are possible. But by the rules of search, flow through arcs in situation (IV) will be exactly at the original upper limit. Flow through arcs in situation (II) must be at the lower limit.

The price adjustment will be added at the initial nodes of these arcs. Thus the situation number can only hold steady or decrease. A situation (IV) arc will move either to situation (III) or else to situation (II). Note again that a situation (III) arc so created will be flowing at its upper limit.

A situation (III) arc will either stay the same or else move to situation (II). Whether a newly-created situation (II) arc was originally in situation (III) or in situation (IV), it will contain a flow at the original upper limit, and thus be eligible in general for inclusion in a breakthrough path.

A situation (II) arc will be moved to situation (I). Note that such arcs must be flowing at their lower limits. A situation (I) arc will remain so.

Thus, as far as the inward-bound set of arcs is concerned, our price adjustment must be the minimum quantity such that at least one of the following events occurs:

- a situation (IV) arc moves to situation (II);
- a situation (III) arc moves to situation (II);

If this adjustment is greater than the one chosen for the outward-bound arcs as described above, the latter quantity will be used.

When the prices at the unmarked nodes have been adjusted, preparation is made for a resumption of the search for breakthroughs. As before, all nodes revert to an unscanned status. The "reached" status and the associated stored information is retained only for those nodes marked as belonging to an infinite breakthrough path. The algorithm is recommenced, starting with the search for an infinite breakthrough.

Note that, throughout the algorithm, capacity expansion will only occur in outward-bound arcs classifiable in situation (IV). In creating a situation (IV) arc we always choose the arc requiring the minimum possible price adjustment. This is equivalent in fact to adding to a path that arc involving the cheapest possible unit expansion cost, given that expansion is necessary.

The optimal expansion routine is implemented in subroutine BUDGET of NETPLAN. The details of the algorithm are documented in Flow Chart D.

Use of the Package

The output resulting from the use of NETPLAN to process a test case on the SIGMA7 system is displayed in the pages following Flow Chart D. Input from the user is underlined for clarity. As passed to the customer, NETPLAN is stored in the file of that name with password the single letter K in order to prevent accidental alteration or deletion. The first step in using the package, given that a load module was not previously stored, is its compilation. As can be seen, this is accomplished by assignment of the file to M:SI followed by a call to the FORTRAN compiler.

In the particular test case shown, only binary output as an object program was desired. Failure to specify any options will result in a listing some 940 statements long -- a rather lengthy process on a teletype! As can be seen, NETPLAN consists of eight routines, each compiled separately.

Following compilation, a call upon the loader prepares the object program for execution. In the illustrated test case, the default temporary file was used for the object module, so that no file name had to be specified. No special options are required of the loader.

When the loader prompted with F: , it was necessary to specify the FORTRAN unit numbers used within the program. FORTRAN unit 1 is used for some of the input, unit 105 for the rest. Which items come in through which unit number is shown in Flow Chart A. All output is through unit 3. Units 1 and 3 are here assigned to the teletype through failure to specify file names for them. Unit 105 is one of the default unit numbers of the FORTRAN system; it need not be assigned explicitly, but merely by means of a carriage return after the other assignments have been completed.

Neither unit 1 nor unit 3 need necessarily be assigned to the teletype. Unit 1 can be assigned to a disk file through the assignment

1= filename, IN

while unit 3 can write to file with the assignment

3= filename, OUT

The separation of input between units 1 and 105 was done deliberately with this application in mind. It must be confessed that file output was not considered in the program design; it would probably be desirable to effect a separtion of output units also if file output is wanted.

Following input/output unit assignment, the loader reports the severity level of errors found (which should be zero). It then asks if the user wishes to execute the program. A response of the letter Y obtains execution. The teletype comes back with an introductory sentence, and then asks if the user wants input instructions suppressed. The user should note that, here and elsewhere in the program, only the first character typed back in a "YES" or "NO" response is tested to see what is wanted. Furthermore, if the first character is not a "Y", the answer is assumed to be "NO". In particular, then, the user should not precede a "YES" answer by any spaces.

If the user does wish suppression of input instructions, every line marked with a little arrow in the left margin will be omitted from the output. This will speed up use of the package, and was designed particularly for file input. The user will, however, need to be aware of what information is wanted of him whenever a question mark is printed on the teletype. In particular, all input indicated by an asterisk in the left margin is read through FORTRAN unit 105, and thus must be typed in regardless of whether file input is being used. Since the request for some of this information only occurs given certain prior answers, it is advisable to keep Flow Chart A handy if print suppression is called for.

Following this first query of the user, the program will continue to introduce itself if print suppression is not asked for. The user will then be asked if he wants to use the flow optimization facility provided in subroutine NETFLO, the logic of which is shown in Flow Chart C. The answer called for here is either "YES" or "NO".

If and only if the answer was "YES", the next input item shown in this test case will be called for. The user, in calling for flow optimization, may wish to specify flow from source to sink, wishing the program to minimize total transportation cost at this flow rate. In this case he should answer with a "1". Not shown in the example is the question that would follow the input of arc data given such an entry. This missing question would be a demand for the specific value of the source-sink flow to be achieved. The answer is read on unit 1.

On the other hand, as in the example case, one can answer the present with a "0". In this case flow from source to sink will be maximized. Then, if alternative patterns will achieve this flow, that pattern will be chosen which minimizes total transport cost. The subsequent question described for an entry of "1" will not appear in this case.

The next question asked of the user is whether he wishes to make use of the network expansion option offered in subroutine BUDGET, of which the logic is described by Flow Chart D. Here again a "YES" or "NO" answer is called for.

The next question shown in the example will only appear if the answer to the previous question was "YES". The user must answer with a "0" if the network expansion will be halted when a budget to be input presently by him is exhausted. If, on the other hand, the user wants the network expanded until the flow from source to sink has reached a level which he will input, he should answer with a "1".

The user is now asked how many nodes are in the network. If the answer is less than two an error message will be printed and the user will be asked to re-type the datum. When the number of nodes has been successfully read in, the user will be asked for the number of arcs. This number excludes any return arc from sink to source. Such an arc will be added within the program for its own purposes, but must not be given by the user. Again, if the number of arcs given is less than one, the user will be asked to re-type his response.

When the program has read in the number of nodes and arcs, it calculates its memory requirements, which vary with these two quantities. If more storage is called for than was provided, an error message will be printed, indicating the corrective action to take. The program will then halt, since the corrections involve changing FORTRAN source statements within the program. For details on memory calculations see the section "Programming Notes" under the sub-heading "Main Program".

The above input was all read in by unit 105 --- that is, over the user's teletype. The next set of input is read from unit 1, and so may, if the user wishes, be read from a disk file as described above.

The general rules for input of these succeeding items are as follows:

(I) All numerical data should be in integer form (no decimal place). This is an essential rather than formal restriction; if the user tries to enter numbers with decimal fractions, rounding to the nearest integer will be performed during input. Integral data is necessary to ensure convergence of the various algorithms used within a finite number of steps. If necessary, the user can scale his input to avoid decimal fractions (e.g., multiply all flows, or all transportation costs, or all expansion costs, by the same suitable power of 10).

(II) Where more than one datum is to be entered on a line, a particular numerical field is terminated by the first comma found, or by the first blank following one or more digits. Blanks preceding a set of digits are ignored. One note: adjacent commas terminate one numerical field (with the first comma), and cause the next number to be read as a zero.

(III) A carriage return ends a given line of input. Any numerical items not yet entered take on a value of zero. Alphanumeric items are correspondingly read as blanks.

(IV) Numeral values should be given in nine or fewer digits.

The information next asked of the user in the identification number and names of the nodes of the network. The user may not use any node reference numbers, in subsequently describing arcs, which were not listed during this step. Moreover, every node listed except the source and the sink must have at least one arc entering it and at least one arc leaving it. The source node may only initiate arcs, while the sink node may only terminate them.

The user is first asked for the reference number and name of the source node. Then similar data is asked for the sink. Finally, the remaining nodes are to be listed, one per line, in any order whatsoever. It is most important that the source node come first, followed by the sink node. Otherwise the program will halt due to apparent errors during the checking of feasibility.

In giving the identifying information, the user should remember that the reference number is terminated by the first blank or comma following it. The name of the node is then read as the next twelve characters following the terminating digit or comma. Thus in the example, due to the form of input, the first character of each name is a blank.

If the user inputs reference numbers which are more than nine digits long, he can expect trouble. Characters in excess of twelve for the names will be ignored; only the first twelve characters will be used to identify nodes on subsequent output.

Following the input of the node identities, the user must describe as many arcs as he has said will be entered. Again, the user must refer only to nodes described previously in defining these arcs. No arc may enter the source node or leave the sink.

All input required for the arcs is numeric. The user must give, in this order, the following items for each arc:

- reference number of initial node
- reference number of terminal node
- lower limit of flow in the arc
- upper limit of flow in the arc (prior to any expansion)
- transportation cost per unit flow through the arc
- investment required per unit additional capacity in the arc
- an initial estimate of flow through the arc. This latter is just to get the routine started; any number will do. If, however, the user is analyzing a series of similar cases, the flows output for one case, if used as input to the next, may save some computation time.

One and only one line should be used to describe each arc.

The rules for termination of numerical fields apply as described above. Within this framework, input format is quite flexible. The program does, unless printing is suppressed, provide a set of column headings. The user may find it useful to place his data under these headings as was done in the example. Note that the user must resist the tempation to supply an arc number; this was done by the program.

The entries "XPORT" and "ADD'TL" beneath the general heading "COST PER UNIT" are abbreviations of "transportation" and "additional" respectively. The latter refers to the investment cost of expansion. Note the correction of a mis-typed cost of transportation for arc 6. This is achieved by hitting the "ESCAPE" and then the "RUBOUT" keys on the teletype, once for each character to be deleted. The deletion of a character by this pair of keys is marked as shown, by the backward arrow. The user can also delete the whole of a line at any point prior to carriage return by hitting the "ESCAPE" key followed by the key for the letter "X".

As the data for each arc is entered, it is checked to see that the lower limit given does not exceed the upper limit. If this error does occur, the user will be informed of the offending arc number, the initial and terminal node numbers, and the limits in question. He will then be asked to type in correct valued for lower and upper limit respectively. Both valued are to be entered on the same line. The usual rules for numeric input apply.

The next item demanded of the user is only needed if he wishes cost minimization in the existing network subject to a given flow from source to sink. In this case, as mentioned above, the user must supply the value of the flow at the present point in the program. As with other requests for input data, only a question mark will be printed if print suppression is in effect.

Except for one item, the remaining responses from the user will be read via unit 105 -- that is, the teletype. The user is now asked if he wishes to review his data. The answer should be "YES" or "NO". If it is affirmative, the node identification and arc data just read in will be displayed. Also shown will be an artificial return arc which was created in the interim. The review facility is particularly of use in providing an otherwise-absent record of input coming in from a disk file.

Next, the user will be given the opportunity to correct any arc data which may have been entered incorrectly. The printed output introducing this is fairly self-explanatory. The same heading is printed to aid the user as was printed for the original entry of arc data (if print suppression was not in effect).

The user must first give the number of an arc to be corrected in response to the question "SEQUENCE NUMBER?". This is the same number which prompted the user during the original input of data for that arc. The arc number also constitutes part of the information printed during the "review of data" earlier in the program.

If the input arc number is zero or negative, arc corrections are assumed to be completed and the program moves on. If the arc number given is greater than the number of arcs read in, the user is asked to re-enter the value. Thus the user cannot adjust data created by the program for the return arc. Finally, if the arc number is within bounds, the user is requested to enter the arc data. He must enter all seven items for the arc on the next line. All remarks on format and content which applied to the original arc data input also apply here.

Following this operation, the program proceeds to check feasibility of the network. The two messages displayed in the example case, regarding the fact that flows are conservative and that a feasible flow was achieved, represent an ideal. It is also possible, without an error being present, to get a message

"*** INITIAL FLOWS NON-CONSERVATIVE --- PROCEEDING TO RECTIFY ***".

This simply means that the flows supplied for the arcs by the user (last input item for each arc) did not satisfy certain balancing conditions. Some of these flows must be re-computed by the program before it can proceed further.

Prior to any message regarding the conservation of flows, the user may be informed that no arc enters or that no arc leaves a certain node. This is generally indicative of an omission on the user's part. If it is not, the data can be rendered palatable to the program through the addition of the necessary dummy arc, possessing zero upper and lower limits. This will only work provided the lower limits on the other arcs adjacent to that node are all zero. In any event, the program will proceed little further until the situation is cleared up.

If the user gets a message about flows being non-conservative, he may also get a further message listing a set of nodes which cannot be reached from the source. Again the program will refuse to proceed much further until the necessary connections are made. As before, dummy arcs may be added if necessary, but it must be possible to satisfy the lower limit conditions on arcs adjacent to the listed nodes.

If none of these error messages occurs, the user will eventually receive the message that "CURRENT FLOWS ARE CONSERVATIVE ...". Then the program proceeds to see if the upper and lower limits on the arcs can indeed all be satisfied at once. If so, the message "FEASIBLE FLOW PATTERN ACHIEVED" is printed. Otherwise the status of a certain arc is printed; this is the arc the program was working on when it found that not all limits were simultaneously satisfiable.

What such a message means is that there is some set of nodes into which, taken as a group, more flow is forced by arc lower limits than can escape (because of the upper limits). The determization of this set or its complement from program output is described in the section "Derivation of Feasible Flows".

After the program has succeeded or failed to create a feasible set of flows, the user is asked whether he wishes to view the resulting flow pattern. The answer should be a "YES" or a "NO". A review of a feasible flow pattern may be of interest for its own sake (e.g. network synthesis without regard to economics). If, on the other hand, the program has informed the user that flow in some arc cannot be corrected, knowledge of the current set of flows is a necessity if the offending set of nodes is to be determined. Thus in this case the user should answer "YES" if he wants to render his network feasible for another attempt. The output generated will be an "Arc Flow Report" as described below.

If some error was discovered and described during the check of feasibility just described, the program considers the present case terminated at this point. The user will be asked if he wants to run a new case, either with the same options previously specified or with a new set of options. In any event his data must be re-entered from the beginning. Unit 1 is rewound; if this is a disk file and remains unaltered, the same data read previously will be read again.

This makes little sense. What can the user do? One way in which he save himself the overhead of re-loading for disk input is suggested as follows. He should first escape to the system monitor when asked if he wishes to re-run. This is done by hitting the "ESCAPE" key twice in succession. When asked if he wishes to proceed, he should answer with an "N".

Next, he should save the current status of core memory as a disk file, using the monitor SAVE command. Then all necessary corrections to the data on the input file should be made, using the system editor.

Now the user can use the monitor RESTORE command, followed by a PROCEED (again a monitor instruction). Then he should answer the original question regarding his desire to re-run; the question was left unanswered by the escape to the monitor. From this point on the program will proceed as before.

If, on the other hand, there were no difficulties during the generation of feasible flows, the next message received will depend upon which progressing options were selected by the user. In the example case, optimization of the existing network was requested. Displayed therefore are the normal two messages, one upon entry and the other upon exit from the optimization routine. These and the subsequent "Arc Flow Report" will be absent if flow optimization was not requested.

The "Arc Flow Report" itself is fairly self-evident in content. One arc is described per line of the report. The nodes initiating and terminating the arcs are listed by name. Next come the lower and upper limits on flow in each arc, as given originally by the user. Following this, under the heading "CURRENT", is given the flow in this arc as per the optimal flow pattern. The final column of the report gives the unit transportation cost through the arc, as read in.

The arcs are listed in order of increasing internal reference number of initial node. For the user, the chief significance of this is that arcs leaving the source node are listed first. In general, all arcs leaving the same node come together in the list.

The last arc given is the return arc, from sink to source, which was created by the program itself. The lower and upper bounds and unit transportation cost were set to fulfil the user's specified objectives. Current flow in this arc is equal to the total flow from source to sink achieved during optimization.

Beneath the arc list are two summary lines. The first line gives the total transportation cost incurred in the network. The second reports the total source-sink flow achieved; this must agree with the current flow in the return arc. One further figure which is not shown, but may be of interest, is the average cost per unit flow from source to sink, obtained by dividing total cost by total flow. On this example it is 120/30 or 4 units.

The remainder of the example case printout, except for the very last line, will only appear if network expansion is requested. Furthermore, as indicated by the marginal arrows, the three lines following the "Arc Flow Report" and requesting more input will not appear if print suppression is in effect. These lines request that the user specify what the value of the limit to network expansion should be. If the user responded "0" when the nature of this limit was requested, the user's answer will be in units of cost. If he responded "1", his answer is in units of total throughput from source to sink desired by him. In the example, expansion up to a total investment of 100 units has been requested.

The line following the user's response (which was read on unit 1, note) is just a progress report. Following this, the cost curve for capacity expansion is traced out, up to the point where the expansion limit has been reached. What is given is the total cumulative investment in added arc capacity required to achieve the given rate of flow from source to sink. The first point given always involves zero investment; this is the maximum flow possible from source to sink in the original network, without expansion of arc capacities. Note that the value of 30 given in the example agrees with the value obtained from the flow optimization routine.

Succeeding points mark breakpoints in a piecewise linear curve of cumulative required investment versus achieved total flow from source to sink. Thus the second pair of values of the example case indicate that a total of 30 cost units would have to be spent on arc capacity expansion in order to obtain a total of 35 units of flow from source to sink. This amounts to a rate of 6 units of cost per unit of flow in the interval from 30 to 35 flow units. Thus, for instance, cumulative investment needed to achieve 32 units of flow would be 12 units of cost.

A similar linear interpolation holds between succeeding pairs of points. Thus to achieve 37 units of flow one would have to invest a total of $30 + (70-30)/(40-35) \times (37-35)$ or 46 units of cost in expanding the original network. The final answer may either be just such an interpolation, or may, as in our example, be an extrapolation beyond the last point given. In the case of an extrapolation, the unit investment required per unit flow since the last breakpoint will apply no matter how much additional flow is desired. Thus in the example the marginal cost for all added flow will be $(100-70)/(42.3-40)$ or 13 units of cost per unit added flow.

Following the notification that the limit of the expansion has been reached, the user receives an "Arc Capacity Expansion Report". Again the arcs of the network are described, one arc per line, arranged in the same order as in the "Arc Flow Report".

First are given the initial and terminal nodes of the arc, by name. Next comes the original upper limit on flow through the arc. Following this is the flow through the arc as required for the higher throughput from source to sink. Note that this may be non-integral due to interpolation.

The flow in any given arc after the network capacity has been expanded may be higher, lower, or the same as it was in the original network. If it has been increased beyond the original upper limit, the incremental capacity will be displayed in the next column, entitled "CAP. CHANGE". This capacity change is costed at the unit rate shown in the following column; these unit rates were given by the user. Finally, the total investment in that arc is the last item of the line.

Following the arc list, the bottom two lines of the report summarize the total investment cost and the throughput from source to sink achieved for it. Note in the example that the total outlay of 100 cost units matches the limit read in. Note also that the program-created return arc has been discarded for the expansion process, and is not reported.

The last line of this example case shows the normal termination of one run of the program. Not shown are two further questions asked of the user. First he is asked if he wishes to re-run with the same options chosen previously. He should answer "YES" or "NO". If he answers affirmatively, he chooses to retain his previous choices for:

- print suppression
- flow optimization, including, if it was called for, the choice of specifying the throughput or having it maximized
- network capacity expansion, including, if it was called for, the choice of a budgetary or a throughput limit.

If the user does not wish to re-run with the same options as before, he is asked if he wishes to re-run with new choices. Again he should answer "YES" or "NO". If the answer is "YES", the program will begin again from the very beginning; otherwise, it will exit to the monitor.

Any re-run will require that all data on the network be read in again. This includes the specification of number of nodes and number of arcs. Unit 1 is rewound prior to any re-run. Thus it is important to note that if input was from disk file, the same file will be read over again for a re-run. Earlier in this section suggestions were made as to how to change data on file for a re-run.

Programming Notes

(a) General

It is strongly advised that a programmer intending to modify NETPLAN read all of this section before doing so. Flow Chart A gives the truest picture of the activities of the NETPLAN package. The other three flow charts were intended more to illustrate the working of particular algorithms. While the logic thus shown parallels that of the actual program, no mention is made of interim or final reports or interfaces with other subroutines.

Wherever the programmer reads "HALT" in these flow charts, he should understand that those other activities instead will take place. The only true stop in the program is in the main program following the user's decision not to re-run.

Note that all program variables are integer unless defined otherwise.

(b) The Main Program

Storage requirements for NETPLAN consist of two parts. One part is fixed, and depends upon the Sigma 7 system as well as program length; the extent of this part has not been determined. The other part varies in extent with the number of arcs and nodes read into the program. The size of this portion of storage, in words of core, is

8 + 8 times the number of arcs
+ 9 times the number of nodes

To this must further be added the number of arcs if capacity expansion has been requested.

All of this variable storage has been placed into the main program array STORE. At present, STORE has been given 1500 words. This is enough to handle, for example, a problem involving 30 nodes and 130 arcs.

If more storage is required, the programmer must change two values in the main program: the dimension of STORE and the value of the variable NWORDS. The statement assigning the latter follows immediately after the

DIMENSION statement. NWORDS is used to check the adequacy of storage provided after the number of nodes and arcs has been read in, and should always equal the dimension of STORE.

The logic of the main program is shown in Flow Chart A. Its primary tasks are to receive some input and to allocate memory from the array STORE to the various arrays needed by the rest of the program routines.

The allocation of STORE proceeds by means of calculations of the starting addresses within STORE of the constituent arrays. The variables used to denote these addresses have the same name as the arrays which correspond to them in all other routines of the program.

The first seven arrays in STORE will contain data read in and in one case modified, for each arc of the network. The length of each of these arrays is equal to the input number of arcs, plus one as a provision for a program-created return arc. The elements of these seven arrays are identified as follows:

FROM	(I)--	number of the initial node of arc I
TO	(I)--	number of the terminal node " " "
LOW	(I)--	lower limit of flow in arc I
HIGH	(I)--	upper limit " " " " "
TOLL	(I)--	unit transportation cost through arc I
ECOST	(I)--	cost per added unit of capacity in arc I
NOW	(I)--	current flow through arc I

All of these arrays are input items. Only the last array is modified by the program, although the order of the elements of all the arrays will in general be changed (see subroutine PRIME, this section). It is essential to the workings of the program that all of these seven arrays be contiguous in storage, and that FROM be the first array. Outside of this, there is no significance to the ordering of the seven arrays in memory.

Following these seven input arrays come three others which are used subsequently to relate nodes to the arcs they initiate and terminate. They are:

INLST --- of length the input number of arcs plus one, this is a list of arc numbers sorted in increasing value of terminal node internal reference number. By construction, the return arc will come first.

(See subroutine WORKER, this section, for a description of node internal reference numbers)

OUTPT --- of length the input number of nodes, this gives for each node the number of the first arc described by the seven arrays FROM...NOW which is initiated by that node.

INPT --- of length the input number of nodes, this gives for each node the address in INLST of the first arc which is terminated by that node.

All three of these lists are created in subroutine PRIME. "Arc number" here refers to the index I of that arc in the arrays FROM...NOW. Note that this index will not generally be the same as it was upon input, because of sorting which occurs in PRIME.

The next two arrays, LABLES and EPSILS, are each of length the number of nodes. They are used to store information during the operation of the various algorithms of the package. As their meaning varies from one part to another of the program, it will receive separate comment in various of the succeeding sub-sections.

The next two arrays store node description data given by the user. They are NOMBRE and NOM. NOMBRE is of length the number of nodes; NOMBRE (I) is the user's reference number for the node internally referred to by the number I. NOM is of effective dimensioning (3, number of nodes). It contains the alphanumeric twelve-character name of each node, and is also indexed by the internal node reference number. If the user is hard-up for storage, elimination of NOM and NOMBRE, with all attendant modica-cations to input and output logic, would appear to be a quick way to gain some.

The next array in STORE is PI. This is of length the number of nodes, and will contain the node market prices needed by the optimizing algorithms. The final array, FSAVE, is needed only in subroutine BUDGET to save old flow patterns. Its length is equal to the number of arcs read in. It was placed last in STORE so that it could be omitted if not needed, releasing that amount of storage.

One other function of the main program is to initialize all of these arrays to zero prior to any input. This is of particular value only for the array PI, which is not otherwise initialized before use in the flow optimization routine NETFLO. It must be confessed the initialization of all of the other arrays was done just on principle, since they either do not require this treatment or else are re-initialized elsewhere.

(c) Subroutine WORKER

Subroutine WORKER has two functions: to accept the remainder of the input, and to set up and call upon the various processing routines. The basic flow of logic of WORKER is shown in Flow Chart A.

The starting addresses of the constituent arrays of STORE are passed through the argument list of WORKER as array formal arguments. Thus WORKER itself deals with the individual arrays, in ignorance of their derivation from STORE.

The first task of WORKER is to read the node identification data into the arrays NOMBRE and NOM and assign internal node reference numbers. The source node is read first, and is assigned the internal number 1. The sink node, read next, is given a reference number equal to the number of nodes to be read. Thus the sink node will be described by the last elements of the arrays INPT, OUTPT, EPSILS, LABLES, NOM, NOMBRE, and PI. All remaining nodes are numbered consecutively in the order read, starting with the number 2.

The arcs are initially stored in the order read in the seven input arrays FROM...NOW. The node reference numbers given in FROM and TO are converted to their internal correspondants, using the information in array NOMBRE. Conversion back to external values is carried out whenever there is communication with the user, as in the review of data or in various error messages.

(d) Subroutine ARCOUT

Subroutine ARCOUT is called upon at two places in WORKER to produce the "Arc Flow Report". This report was described in the previous section of this document, and is illustrated in the example printout following Flow Chart D.

(e) Subroutine PRIME

Subroutine PRIME is responsible, jointly with subroutine NETFLO which it calls, for the checking of network feasibility and generation of a feasible flow pattern. PRIME calls upon subroutine SORTS to sort arc data, and then prepares the lists in arrays INLST, INPT, and OUTPT, as a preliminary step in this process.

Subroutine SORTS assumes that there are seven arc data arrays, all of length the number of input arcs plus one, and all adjacent in storage. It sorts the elements of these arrays so that the arc descriptions are stored in order of increasing value of the numbers in the first array of the set. These assumptions explain the restriction on storage arrangement declared in the sub-section on the main program above.

Since the first array of the set is FROM, we end up with the arcs ordered by increasing initial node internal reference number. This means that all arcs leaving the source come first, and the return arc comes last. The construction and use of the arrays INLST, INPT, and OUTPT are predicated upon this ordering, as are various other pieces of logic throughout the program. Let the programmer therefore beware of changing this arrangement!

The contents of INLST, INPT, and OUTPT have been described already in the main program sub-section. They are used to speed up the orderly examination of all arcs leaving or entering a particular node--a process which is repeated frequently in the various algorithms of PRIME, NETFLO, and BUDGET.

The logic for derivation of conservative flows by the generation of a tree spanning the network has been described previously, in the section "The Algorithms: Preliminary Discussion."
Within this procedure, the variable CT counts the count of the number of nodes labelled so far. When this equals the total number of nodes, the generation of the tree is complete.

Also in this section of the program, the array LABLES contains the number of the arc linking each node to the next lower level of the tree. (In this sense, the source node is at the lowest level, since it is the root of the tree). To this arc number is assigned a minus sign when it is first stored. This minus sign is an indication that the arcs leaving that node have not yet been examined to see if they will extend the tree. The minus sign is removed when examination begins, so that no node will be processed twice.

The array EPSILS contains a count for each node of the number of arcs leaving it which belong to the tree. Thus a node for which EPSILS is zero after the tree has been generated is the terminal node of one arc of the tree, but has no arcs of the tree leaving it.

Once a tree has been generated, flows are calculated for the arcs of the tree in terms of the flows for other arcs of the network. Calculations are begun with the highest-level nodes---those for which EPSILS is zero. For these only one adjacent arc will have an undefined flow. This is determined quickly by the requirement that flow entering a node equal flow leaving it.

When an arc flow has been computed in this way, the count of outgoing arcs in EPSILS for the lower-level node initiating the newly-determined arc is decreased by 1. The value of EPSILS for the higher-level node entered by that arc is set to -1 so that it will not be processed again. Thus the tips of the tree, where EPSILS has value 0, are steadily moved back to the source node.

When only the source node remains to be processed, all arc flows have actually been determined. It would be an error to process the source node. To prevent this within the context of the logic, EPSILS (1) is initialized to -1 before the tree is generated; all other element of EPSILS are initially 0. The scanning of nodes for zero values of EPSILS during flow determination proceeds from the highest node number to the lowest, so that the source node is never considered before EPSILS (1) has once again been reduced to -1.

PRIME completes the checking of feasibility and generation of feasible flows by calling upon NETFLO with the variable TASK set to 1. The arguments PI and TOLL of NETFLO have no relevance under these circumstances, and are just dummy variables. If NETFLO were modified to use elements of either of these arrays with TASK set to 1, errors would result unless they were first passed to and dimensioned in PRIME. Again, let the programmer beware!

(f) Subroutine SORTS

Subroutine SORTS is a general sorting routine, capable of sorting a number of adjacent arrays keyed to one array of the set. The comments preceding the listing of this subroutine provide adequate documentation on its general use. SORTS is called once, by PRIME.

(g) Subroutine NETFLO

A general view of the role of NETFLO is given in Flow Chart A. Flow Charts B and C show the internal logic of NETFLO with the variable TASK set equal to 1 and to 2 respectively. With TASK equal to 1, NETFLO generates a feasible flow pattern if this is possible, or else gives an indication of where the bottlenecks lie. With TASK equal to 2, NETFLO finds least-cost maximal flow patterns. Note the caveat given in PRIME against modifying NETFLO to use array PI or TOLL with TASK equal to 1.

The arrays LABLES and EPSILS are again used to store data during the processing.. A node is marked "reached" if it has a non-zero value in LABLES. It is marked "scanned" if its value in EPSILS is set equal to INF (or infinity, set to $2^{31} - 1$ at the start of the routine).

The number of the arc reaching a node is stored in LABLES. Furthermore, the direction of flow change along that arc is indicated by appending a plus or minus sign as applicable to that arc number. Thus the entries in LABLES provide three separate pieces of information.

When a node is reached, the maximum permitted flow change for that node is stored in EPSILS. This value is only needed until the node has been scanned, when it is replaced with INF as described above. Thus EPSILS serves two separate purposes. Both EPSILS and LABLES are set to zero at the start of a search.

Note deviations in the programming from points of view expressed in earlier sections of this document in describing the two NETFLO algorithms. Thus, for instance, the economic decision variable ABAR is the negative of the profit incentive in an arc discussed earlier. The program usage is more consistent with the original Ford-Fulkerson papers.

(h) Subroutine BUDGET

This subroutine has as purpose the computation of the minimum-cost set of additions to arc capacities to meet a given throughput objective. Conversely, for any given expended budget, the routine will return that set of arc capacity expansions which maximizes throughput. The logic of BUDGET is outlined in Flow Charts A and D.

A number of variables specific to BUDGET may be of special interest to the programmer. First, the variable TYPE is a switch read in from the user as SW6 on page A-3, Flow Chart A. It determines the type of limit on expansion.

Secondly, note that BUDGET, alone of all routines in NETPLAN, defines some real variables. These are used strictly in presenting output from the routine. They are needed because it is necessary in general to interpolate between successive sets of solutions for the final answers.

Possible solutions correspond to those flow patterns and cumulative expenditure levels prevailing when non-breakthroughs immediately follow breakthroughs. The final answer comes from interpolating between two succeeding such events, or by extrapolating beyond the latest one in the case of an infinite breakthrough.

In order to keep track of the situation, the variable LAST is set to zero when a non-breakthrough occurs, and unity when there is a finite breakthrough. LAST is the breakthrough switch mentioned in Flow Chart D. The transition from a value of unity to a value of zero then marks a point where a solution has been generated.

The array FSAVE, with one element for each arc in the network, is used to save the current pattern of flows at such times. Then and at all other times the current flows reside in the array NOW. Corresponding to these flow patterns are the accumulated expenditures for capacity expansion. SPENT contains the current level at all times, while SPENSV contains the level corresponding to the flows in FSAVE. Finally, the related total flows from source to sink are held in FLOW and FLOWSV respectively. Note that FLOW is initially set to the value of the element NOW (ARCS + 1) because this is the value contained in the artificial return arc which was used in previous routines.

The real variables mentioned above are ADD, OUTFLO, ALPHA, CHANGE, and OUTLAY. All are used only at the logical point of output. In order to avoid the necessity of carrying a real array to handle non-integral flows, interpolation or extrapolation is done one arc at a time.

For the programmer, OUTFLO and CHANGE are the variables possibly of most interest. At the point of output, OUTFLO is the flow in the arc being reported. CHANGE is the amount of increase in that arc's capacity, if any, from the original upper limit. OUTFLO is created in one of two ways. In the case of a finite breakthrough, interpolation is between the flows of FSAVE and NOW, using the interpolation factor ALPHA. For an infinite breakthrough, ADD is added to NOW for those arcs previously marked as belonging to the infinite breakthrough path. For other arcs, OUTFLO is equal to the current value of NOW.

Once again, searches are carried out using LABLES and EPSILS to store information in a way which is unique to the routine. The difference from the usage in NETFLO is fairly minor. It is now necessary to mark nodes as having been reached during a scan for infinite breakthrough. This is done by giving EPSILS a value of infinity, as stored again in the variable INF.

This necessitates a new way to mark a node as "scanned", since this was done in NETFLO using INF. Instead, in BUDGET, a node is marked by giving its value in EPSILS a minus sign. The usage of LABLES is the same in BUDGET as in NETFLO; BUDGET tests if a node has been "reached" by noting a non-zero LABLES value.

(i) Subroutine CLEAR

This is the final routine of the program. It is a small service routine called at two different points by BUDGET to prepare EPSILS and LABLES before starting a new search for breakthrough. The actions taken correspond to the top two blocks on page D-4, Flow Chart D, and to the similar two blocks on page D-10.

CLEAR scans all nodes. Those with EPSILS equal to minus INF have that value set to plus INF. Nodes with EPSILS equal to plus INF are left alone. These nodes were reached during a search for infinite breakthrough. LABLES contains for each of them the number of the arc which reached it. All other nodes have LABLES and EPSILS set to zero.

References

1. Flows in Networks, L. R. Ford Jr. and D. R. Fulkerson, Princeton University Press, Princeton,ceton, New Jersey, 1962.

2. D. R. Fulkerson, "Increasing the Capacity of a Network: The Parametric Budget Problem," Man. Sci. 5 (1959), 472-483.

3. A.J. Hoffman, "Some Recent Applications of the Theory of Linear Inequalities to Extremal Combinatorial Analysis," Proc. Symposia on Applied Math. 10 (1960).

```
!ASSIGN M:SI,(FILE,NETPLAN),(PASS,)

!FORTRAN
OPTIONS: BO
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **
** END OF COMPILATION **

!LOAD
ELEMENT FILES:
OPTIONS:
F:1
F:3
F:

SEV.LEV. = 0
XEQ? Y

THE NETWORK FLOW ANALYSIS PACKAGE AWAITS YOUR COMMANDS
DO YOU WISH TYPING OF INPUT INSTRUCTIONS SUPPRESSED
( YES OR NO)
?NO
THREE SERVICES ARE OFFERED:
 -- FEASIBILITY CHECKING OF THE INPUT NETWORK GIVEN THE
    SPECIFIED MINIMUM AND MAXIMUM FLOWS
 -- ADJUSTMENT OF FLOWS IN INDIVIDUAL ARCS TO MINIMIZE
    TOTAL TRANSPORT COST, WITH TOTAL CIRCULATION FROM SOURCE
    TO SINK EITHER SPECIFIED OR TO BE MAXIMIZED
DO YOU WANT THIS
?YES
FOR THIS EXISTING NETWORK OPTIMIZATION, WILL YOU GIVE
TOTAL CIRCULATION (IF SO, TYPE 1), OR IS IT TO BE
FOUND (IF SO, ANSWER 0)
?0
 -- FINALLY, THE PROGRAM WILL DETERMINE THE LEAST-EXPANSION-COST PATTERN
    OF CAPACITY EXPANSION, EITHER TO PROVIDE A GIVEN
    INCREASE IN THROUGHPUT AT LEAST COST, OR ELSE TO
    MAXIMIZE ADDED THROUGHPUT FOR A GIVEN EXPANSION BUDGET
DO YOU WANT THIS
?YES
WILL YOU GIVE THE ULTIMATE THROUGHPUT TO BE OBTAINED
( ANSWER 1), OR THE BUDGET WITHIN WHICH THE EXPANSION WILL BE
HELD (ANSWER 0)
?0
HOW MANY NODES IN THE NETWORK
?5
HOW MANY ARCS IN THE NETWORK
```

Password K in here
was not printed by system

```
?7
NØW GIVE THE FØLLØWING DATA FØR EACH NØDE IN TURN:
   REFERENCE NØ., ALPHANUMERIC NAME
GIVE THIS FIRST FØR THE SØURCE NØDE
?10, VANCØUVER
NØW GIVE DATA FØR THE TERMINAL NØDE
?50, TØRØNTØ
NØW DESCRIBE ALL ØTHER NØDES, ØNE NØDE PER LINE
? 20, EDMØNTØN
?30, CALGARY
?40, WINNIPEG
NØW GIVE DATA FØR EACH ARC ØF THE NETWØRK IN TURN WHEN
YØU ARE PRØMPTED BY THE PRINTING ØF THE ARC SEQUENCE NØ.  FØR
EACH ARC GIVE THE FØLLØWING DATA:
   INITIAL NØDE NØ., ENDING NØDE NØ., MINIMUM FLØW, MAXIMUM
   INITIAL CAPACITY, UNIT TRANSPØRTATIØN CØST, CØST PER UNIT
   ADDED CAPACITY, AND INITIAL ESTIMATED FLØW
N.B.  INSERT ZERØES IN PLACE ØF ANY NØN-APPLICABLE DATA
ARC    FRØM     TØ      MIN.    MAX.    CØST PER UNIT   INIT.
       (NØ.)    (NØ.)   FLØW    FLØW    XPØRT ADD'TL    FLØW

 1:
?       30       40      0       10       3      4        0
 2:
?       10       40      5       15       2      8        0
 3:
?       20       40      0       10       1      5        0
 4:
?       10       20      5       10       3     10        0
 5:
?       10       30      5       10       2      8        0
 6:
?       40       50      5       20      6-2     6        0
 7:
?       30       50      5       15       1      5        0
DØ YØU WISH TØ REVIEW YØUR INPUT
?NØ
DØ YØU WISH TØ CØRRECT ANY ARC DATA?
IF SØ, GIVE THE SEQUENCE NUMBER ØF THE ARC WHEN ASKED
SIGNAL THE END ØF CØRRECTIØNS BY A SEQUENCE NØ. ØF 0
ARC    FRØM     TØ      MIN.    MAX.    CØST PER UNIT   INIT.
       (NØ.)    (NØ.)   FLØW    FLØW    XPØRT ADD'TL    FLØW

SEQUENCE NUMBER?

?0
CURRENT FLØWS ARE CØNSERVATIVE, NØW CHECKING FEASIBILITY
FEASIBLE FLØW PATTERN ACHIEVED
DØ YØU WANT TØ SEE THE FLØWS
?NØ
ENTERING MINIMUM-CØST MAXIMAL FLØW RØUTINE
FLØW PATTERN SUCCESSFULLY ØPTIMIZED
```

ARC FLØW REPØRT

FRØM	TØ	FLØWS IN THIS ARC MINIMUM	MAXIMUM	CURRENT	UNIT TØLL
VANCØUVER	CALGARY	5	10	10	2
VANCØUVER	EDMØNTØN	5	10	5	3
VANCØUVER	WINNIPEG	5	15	15	2
EDMØNTØN	WINNIPEG	-0	10	5	1
CALGARY	TØRØNTØ	5	15	10	1
CALGARY	WINNIPEG	0	10	0	3
WINNIPEG	TØRØNTØ	5	20	20	2
ARTIFICIAL RETURN ARC		0	99999	30	-99999

CURRENT TØTAL TRANSPØRTATIØN CØST IS 120
CURRENT CIRCULATIØN FRØM SØURCE TØ SINK IS 30

YØU WISH THE NETWØRK EXPANDED
PLEASE GIVE THE AMØUNT ØF THE ULTIMATE THRØUGHPUT ØR
EXPANSIØN BUDGET, ACCØRDING TØ WHICH ØPTIØN YØU CHØSE ABØVE
?100

ENTERING MINIMUM-CØST CAPACITY EXPANSIØN RØUTINE
THE FØLLØWING PARAMETRIC DATA MAY BE ØF INTEREST
LINEAR INTERPØLATIØN ØBTAINS BETWEEN SUCCESSIVE PAIRS ØF VALUES.

BUDGET	FLØW
0	30
30	35
70	40

EXPANSIØN CARRIED TØ SPECIFIED LIMIT

ARC CAPACITY EXPANSIØN REPØRT

FRØM	TØ	ØLD MAX	NEW FLØW	CAP. CHANGE	INVESTMENT REQUIRED UNIT	TØTAL
VANCØUVER	CALGARY	10	17.3	7.3	8	58.5
VANCØUVER	EDMØNTØN	10	10.0	.0	10	.0
VANCØUVER	WINNIPEG	15	15.0	.0	8	.0
EDMØNTØN	WINNIPEG	10	10.0	.0	5	.0
CALGARY	TØRØNTØ	15	17.3	2.3	5	11.5
CALGARY	WINNIPEG	10	.0	.0	4	.0
WINNIPEG	TØRØNTØ	20	25.0	5.0	6	30.0

TØTAL INVESTMENT ØUTLAY WAS 100.0
THRØUGHPUT ACHIEVED WAS 42.3
THAT'S ALL ØN THE AGENDA -- C'EST TØUT

EXAMPLE: PROGRAM NETPLAN(,K)

```
!LOGIN: PLANNING,1004S,POLICY
ID= 3
!LOAD
ELEMENT FILES: NETPLANB
OPTIONS:
F:1
F:3
F:

SEV.LEV. = 0
XEQ? Y

THE NETWORK FLOW ANALYSIS PACKAGE AWAITS YOUR COMMANDS
DO YOU WISH TYPING OF INPUT INSTRUCTIONS SUPPRESSED
(YES OR NO)
?NO
THREE SERVICES ARE OFFERED:
 --- FEASIBILITY CHECKING OF THE INPUT NETWORK GIVEN THE
     SPECIFIED MINIMUM AND MAXIMUM FLOWS
 --- ADJUSTMENT OF FLOWS IN INDIVIDUAL ARCS TO MINIMIZE
     TOTAL TRANSPORT COST, WITH TOTAL CIRCULATION FROM SOURCE
     TO SINK EITHER SPECIFIED OR TO BE MAXIMIZED
DO YOU WANT THIS
?YES
FOR THIS EXISTING NETWORK OPTIMIZATION, WILL YOU GIVE
TOTAL CIRCULATION (IF SO, TYPE 1), OR IS IT TO BE
FOUND (IF SO, ANSWER 0)
?0
 --- FINALLY, THE PROGRAM WILL DETERMINE THE LEAST-EXPANSION-COST PATTERN
     OF CAPACITY EXPANSION, EITHER TO PROVIDE A GIVEN
     INCREASE IN THROUGHPUT AT LEAST COST, OR ELSE TO
     MAXIMIZE ADDED THROUGHPUT FOR A GIVEN EXPANSION BUDGET
DO YOU WANT THIS
?YES
WILL YOU GIVE THE ULTIMATE THROUGHPUT TO BE OBTAINED
(ANSWER 1), OR THE BUDGET WITHIN WHICH THE EXPANSION WILL BE
HELD (ANSWER 0)
?0
HOW MANY NODES IN THE NETWORK
?4
HOW MANY ARCS IN THE NETWORK
?5
NOW GIVE THE FOLLOWING DATA FOR EACH NODE IN TURN:
   REFERENCE NO., ALPHANUMERIC NAME
GIVE THIS FIRST FOR THE SOURCE NODE
?1,TORONTO
NOW GIVE DATA FOR THE TERMINAL NODE
?4,QUEBEC
NOW DESCRIBE ALL OTHER NODES, ONE NODE PER LINE
?2,OTTAWA
?3,MONTREAL
```

```
NOW GIVE DATA FOR EACH ARC OF THE NETWORK IN TURN WHEN
YOU ARE PROMPTED BY THE PRINTING OF THE ARC SEQUENCE NO.  FOR
EACH ARC GIVE THE FOLLOWING DATA:
   INITIAL NODE NO., ENDING NODE NO., MINIMUM FLOW, MAXIMUM
   INITIAL CAPACITY, UNIT TRANSPORTATION COST, COST PER UNIT
   ADDED CAPACITY, AND INITIAL ESTIMATED FLOW
N.B.  INSERT ZEROES IN PLACE OF ANY NON-APPLICABLE DATA
ARC      FROM      TO       MIN.     MAX.    COST PER UNIT   INIT.
         (NO.)     (NO.)    FLOW     FLOW    XPORT ADD'TL    FLOW

 1:
?1       1         2        5        15      10    50        0
 2:
?        1         3        0        25      12    40        0
 3:
?        2         3        5        15      20    15        0
 4:
?        2         4        5        10      20    100       0
 5:
?        3         4        0        30      10    40        0
DO YOU WISH TO REVIEW YOUR INPUT
?YES
ARC      FROM      TO       MIN.     MAX.    COST PER UNIT   INIT.
         (NO.)     (NO.)    FLOW     FLOW    XPORT ADD'TL    FLOW

  1         1         2        5       15       10      50        0
  2         1         3        0       25       12      40        0
  3         2         3        5       15       20      15        0
  4         2         4        5       10       20     100        0
  5         3         4        0       30       10      40        0
  6         4         1        0    99999   -99999       0        0


NODE NO.   NAME

       1 TORONTO
       2 OTTAWA
       3 MONTREAL
       4 QUEBEC
DO YOU WISH TO CORRECT ANY ARC DATA?
IF SO,GIVE THE SEQUENCE NUMBER OF THE ARC WHEN ASKED
SIGNAL THE END OF CORRECTIONS BY A SEQUENCE NO. OF 0
ARC      FROM      TO       MIN.     MAX.    COST PER UNIT   INIT.
         (NO.)     (NO.)    FLOW     FLOW    XPORT ADD'TL    FLOW

UEQUENCE NUMBER?

?0
CURRENT FLOWS ARE CONSERVATIVE, NOW CHECKING FEASIBILITY
FEASIBLE FLOW PATTERN ACHIEVED
DO YOU WANT TO SEE THE FLOWS
```

ARC FLOW REPORT

FROM	TO	FLOWS IN THIS ARC MINIMUM	MAXIMUM	CURRENT	UNIT TOLL
TORONTO	MONTREAL	0	25	25	12
TORONTO	OTTAWA	5	15	15	10
OTTAWA	QUEBEC	5	10	10	20
OTTAWA	MONTREAL	5	15	5	20
MONTREAL	QUEBEC	0	30	30	10
ARTIFICIAL RETURN ARC		0	99999	40	-99999

CURRENT TOTAL TRANSPORTATION COST IS 1050
CURRENT CIRCULATION FROM SOURCE TO SINK IS 40

YOU WISH THE NETWORK EXPANDED
PLEASE GIVE THE AMOUNT OF THE ULTIMATE THROUGHPUT OR
EXPANSION BUDGET, ACCORDING TO WHICH OPTION YOU CHOSE ABOVE
?500

ENTERING MINIMUM-COST CAPACITY EXPANSION ROUTINE
THE FOLLOWING PARAMETRIC DATA MAY BE OF INTEREST
LINEAR INTERPOLATION OBTAINS BETWEEN SUCCESSIVE PAIRS OF VALUES

BUDGET	FLOW
0	40
0	40
0	40

EXPANSION CARRIED TO SPECIFIED LIMIT

ARC CAPACITY EXPANSION REPORT

FROM	TO	OLD MAX	NEW FLOW	CAP. CHANGE	INVESTMENT REQUIRED UNIT	TOTAL
TORONTO	MONTREAL	25	31.3	6.2	40	250.0
TORONTO	OTTAWA	15	15.0	.0	50	.0
OTTAWA	QUEBEC	10	10.0	.0	100	.0
OTTAWA	MONTREAL	15	5.0	.0	15	.0
MONTREAL	QUEBEC	30	36.2	6.2	40	250.0

TOTAL INVESTMENT OUTLAY WAS 500.0
THROUGHPUT ACHIEVED WAS 46.2
THAT'S ALL ON THE AGENDA -- C'EST TOUT

DO YOU WANT TO RE-RUN WITH SAME OPTIONS
?NO
DO YOU WANT TO RE-RUN WITH NEW OPTIONS
?NO
STOP 0

!BYE
09/13/'71 09:52
RAD SPACE 0
CPU TIME 0.258
I/O WAIT TIME 0.071
MON SERVICES 0.035

SECTION V

PRESENTATION SLIDES

PRESENTATION

NETWORKS

NETWORK SYNTHESIS

OPTIMAL NETWORK EXPANSION

RELIABILITY

RELIABILITY PREDICTION

PROBABALIST BEHAVIOR OF LARGE SYSTEMS

DECISION MAKING

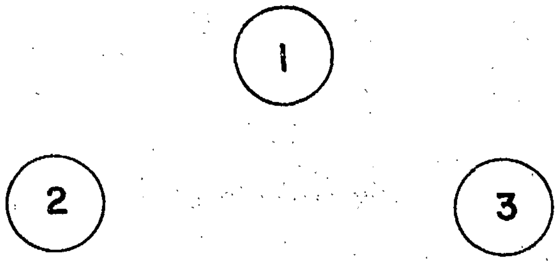
OPTIMAL SEQUENTIAL DECISION MAKING

BEST STRATEGIES FOR MAXIMUM REWARDS
OR MINIMUM LOSSES

NETWORKS

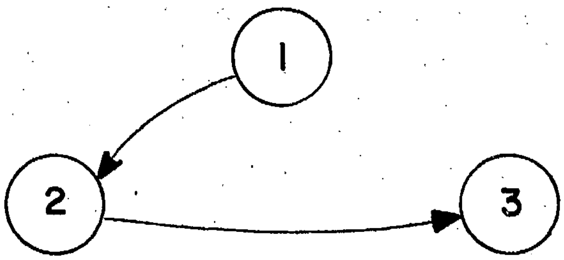
GIVEN:

N - THE NODES OR TERMINALS
T - THE REQUIREMENTS AT THE NODES
K - THE COST AND GEOGRAPHIC CONSTRAINTS



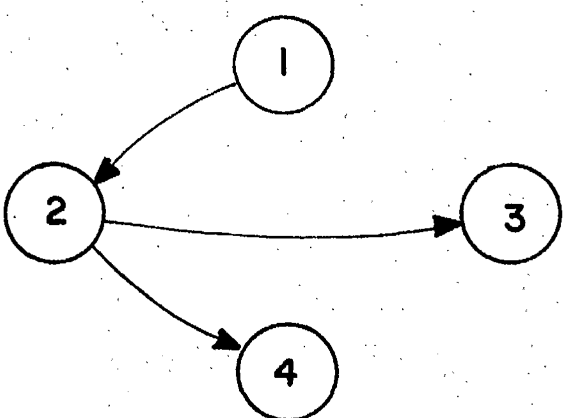
FIND:

R - THE CONNECTIONS (CHANNELS) GIVEN T AND K.

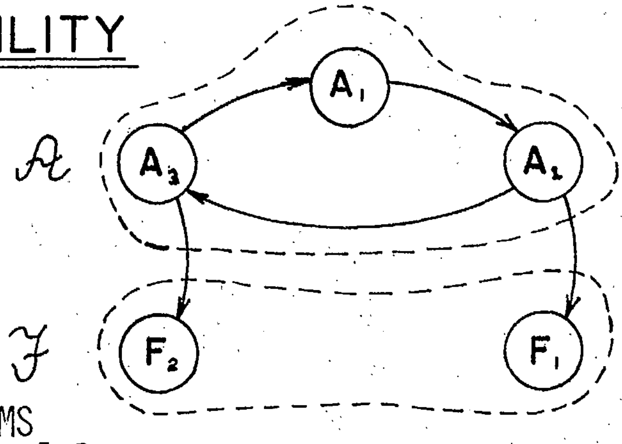


THE OPTIMAL EXPANSION OF THE NETWORK GIVEN

A) ADDITIONAL DEMANDS AT THE NODES
B) NEW NODES WITH NEW DEMANDS
C) A FIXED BUDGET TO BE OPTIMALLY ALLOCATED



RELIABILITY



GIVEN:

A) HOW THE SYSTEM WORKS:
  I) THE ACCEPTABLE STATES
  II) THE FAILURE STATES
B) THE FAILURE AND REPAIR RATES OF THE COMPONENT SUBSYSTEMS

CONSTRUCT THE SYSTEM MATRIX (M):

THE ENTRIES OF THE MATRICES A AND B OF M ARE THE REPAIR AND FAILURE RATES OF THE INDIVIDUAL SYSTEM.

$$[M] = \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ F_1 \\ F_2 \end{array} \begin{array}{c} \begin{array}{ccccc} A_1 & A_2 & A_3 & F_1 & F_2 \end{array} \\ \left[\begin{array}{ccc|cc} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ \hline 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} = \begin{bmatrix} [A] & [B] \\ [0] & [I] \end{bmatrix}$$

PROGRAM FOR RELIABILITY ANALYSIS:

WITH A AND B AS INPUT, THE PROGRAM COMPUTES:

[R(N)] - PROBABILITY THAT SYSTEM IS IN ACCEPTABLE STATE IN $\mathcal{A}$ AT TIME N.
[P(N)] - TRANSITION PROBABILITY FUNCTIONS
[P] - STEADY STATE FAILURE PROBABILITIES

PROGRAM NAME: MAT(.PROB)

MATRIX ANALYSIS

MATRIX ADDITION, MULTIPLICATION

INVERSION OF A MATRIX FUNCTION

FUNCTIONS OF A MATRIX

A) EXPONENTIAL FUNCTIONS

B) POWER FUNCTIONS

PROGRAM NAME: MAT(,PROB)

DECISION MAKING

GIVEN: A) THE VARIOUS ALTERNATIVES = 1, 2, --- N

B) THE POSSIBLE STRATEGIES ASSOCIATED WITH EACH ALTERNATIVE

C) THE TRANSITION PROBABILITIES BETWEEN THESE ALTERNATIVES

D) THE REWARD OR LOSS ASSOCIATED WITH EACH OF THESE ALTERNATIVES

FIND: THE MAXIMUM REWARD OR MINIMUM LOSS FUNCTION AND THE CORRESPONDING STRATEGIES TO BE FOLLOWED::

PROGRAM INPUT:

$$\left\{ \begin{matrix} [P]_i^{d_i} , & \cdots\cdots\cdots , & [P]_N^{d_k} \\ \vdots & & \vdots \\ [R]_i^{d_i} , & \cdots\cdots\cdots , & [R]_N^{d_k} \end{matrix} \right\}$$

PROGRAM OUTPUT:

MAXIMUM REWARD OR MINIMUM LOSS FUNCTION, $\overline{V}$ (N) AND THE CORRESPONDING STRATEGIES WHERE $\overline{V}$ (N) ≡ MAXIMUM REWARD OR MINIMUM LOSS FUNCTION FOR THE FOLLOWING STRATEGY $\overline{D}$ (N) AT TIME N.

PROGRAM NAME: MAX(,MIN)

PROGRAM SHORT1(,K)

I- WHAT DOES THE PROGRAM REQUIRE ?

1- N- THE NUMBER OF NODES

2- T- THE TERMINAL CAPACITY REQUIREMENTS

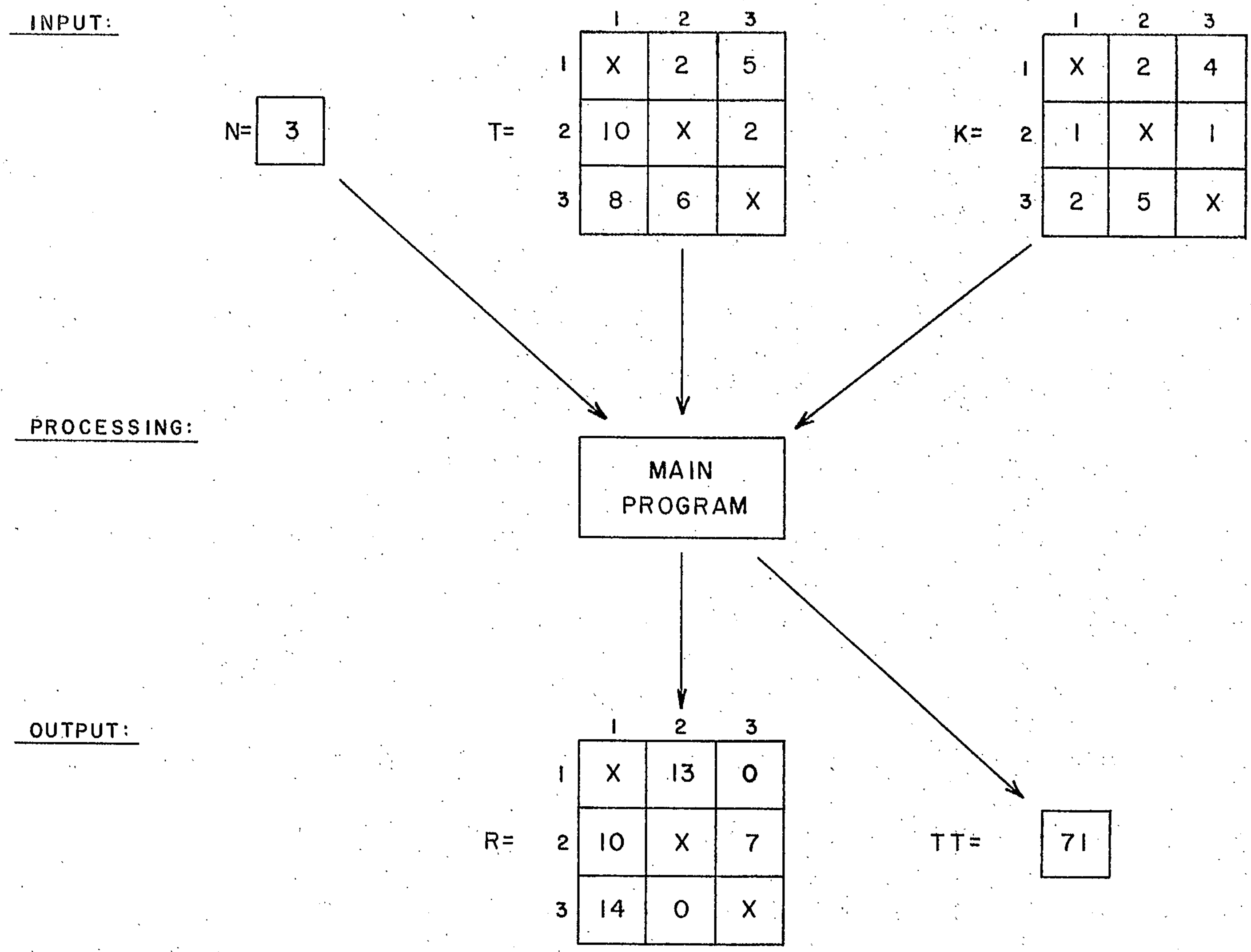
3- K- THE ARC COST CONSTRAINTS

II- WHAT DOES THE PROGRAM DO ?

THE PROGRAM FINDS THE ARC CAPACITIES, R, THAT ARE REQUIRED TO SATISFY ALL THE REQUIREMENTS SIMULTANEOUSLY AT MINIMUM TOTAL NETWORK COST.

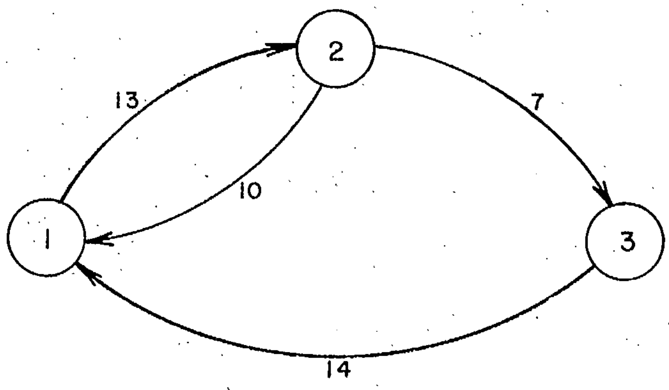
PROGRAM SHORT I (,K)

SYNTHESIS OF SIMULTANEOUS TRANSMISSION NETWORKS



SHORT 1(,K)

THE NETWORK:



PROGRAM SHORT2(,K)

I- WHAT DOES THE PROGRAM REQUIRE ?

1- N- THE NUMBER OF NODES

2- T- THE TERMINAL CAPACITY REQUIREMENTS

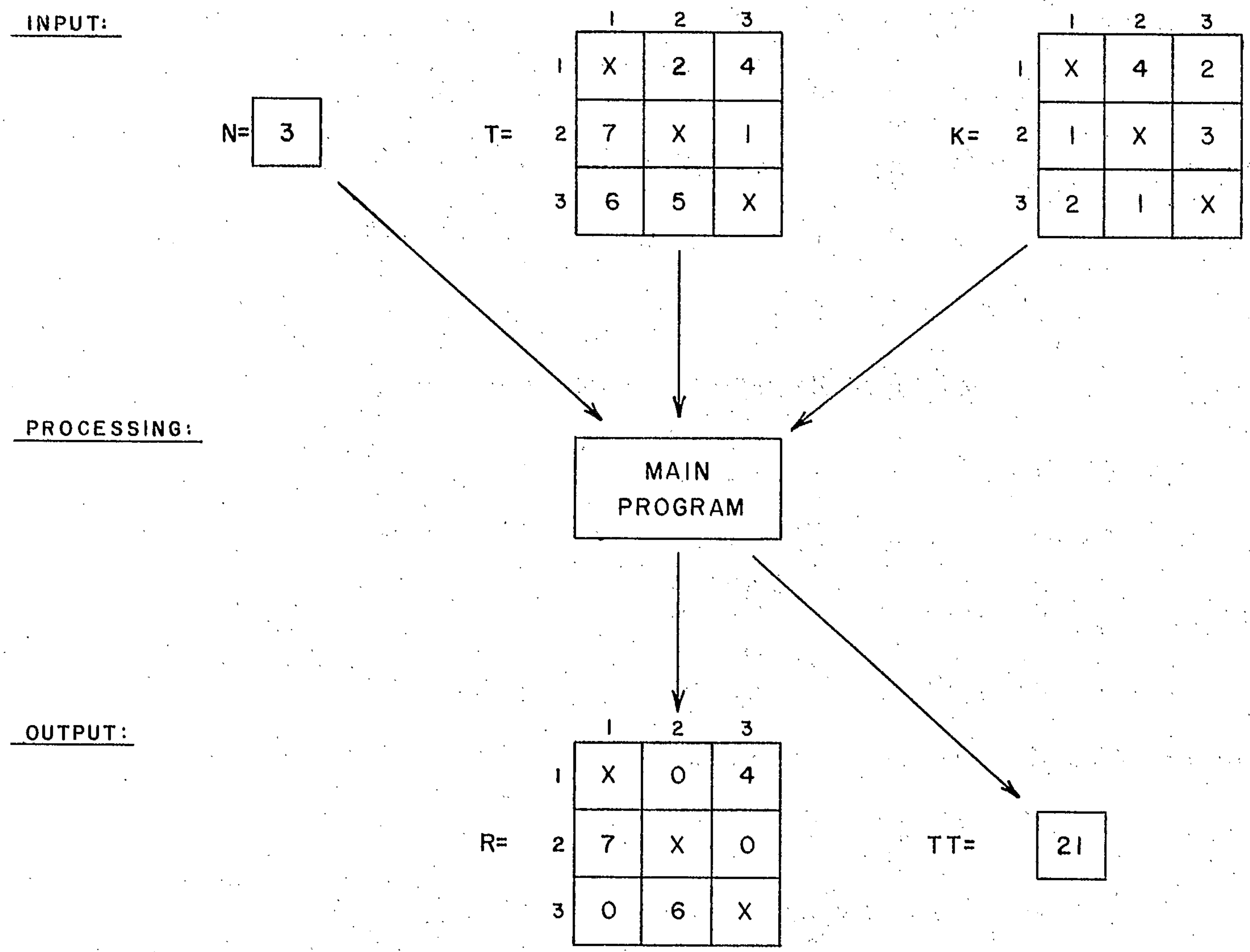
3- K- THE ARC COST CONSTRAINTS

II- WHAT DOES THE PROGRAM DO ?

THE PROGRAM FINDS THE ARC CAPACITIES, R, THAT ARE REQUIRED TO SATISFY ONE TERMINAL PAIR AT A TIME. THIS IS THE TIME-SHARED CONFIGURATION. THE RESULTANT NETWORK IS ONE OF REDUCED COST.

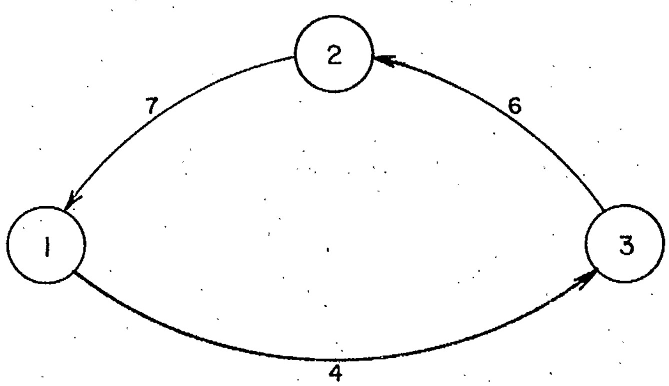
PROGRAM SHORT 2 (,K)

SYNTHESIS OF TIME-SHARED COMMUNICATIONS NETWORKS



SHORT 2(,K)

THE NETWORK:



PROGRAM NETSYM1(,K)

I- WHAT DOES THE PROGRAM REQUIRE ?

1- N- THE NUMBER OF NODES

2- T- THE TERMINAL CAPACITY REQUIREMENTS

3- K- THE ARC COST CONSTRAINTS

II- WHAT DOES THE PROGRAM DO ?

THE PROGRAM SYNTHESIZES A TIME-SHARED COMMUNICATIONS NETWORK IN WHICH THERE ARE CERTAIN TERMINAL CAPACITY REDUNDANCIES IN T. THE ARC CAPACITIES, R, ARE FOUND SUCH THAT THE TERMINAL REQUIREMENTS ARE EXACTLY SATISFIED.

PROGRAM NETSYM I (,K)

REQUIREMENTS ARE EXACTLY SATISFIED

INPUT:

N= 4

T=

	1	2	3	4
1	X	2	2	2
2	3	X	4	6
3	3	7	X	8
4	3	5	4	X

K=

	1	2	3	4
1	X	2	1	2
2	1	X	2	0
3	2	0	X	0
4	2	0	1	X

PROCESSING:

MAIN PROGRAM

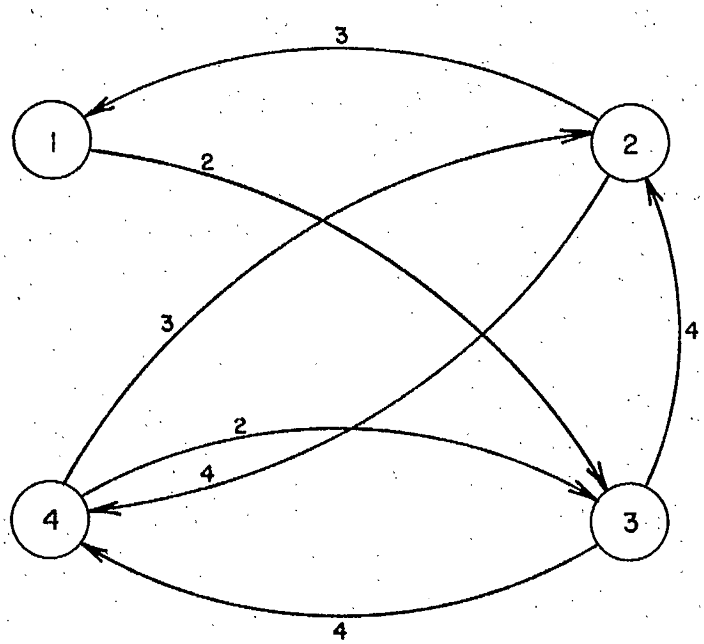
OUTPUT:

R=

	1	2	3	4
1	X	0	2	0
2	3	X	0	4
3	0	4	X	4
4	0	3	2	X

NETSYM1(,K)

THE NETWORK:



PROGRAM NETPLAN(,K)

I- WHAT DOES THE PROGRAM REQUIRE ?

IT REQUIRES A COMMUNICATIONS NETWORK WITH:

1- ARC CAPACITIES i) UPPER
ii) LOWER

2- ARC RENTAL COSTS (COST PER UNIT FLOW)

3- ARC EXPANSION COSTS (COST PER UNIT CAPACITY)

II- WHAT DOES THE PROGRAM DO ?

FOR A GIVEN PAIR OF NODES I AND J IN THE NETWORK THE PROGRAM ACCOMPLISHES (1) AND/OR (2) BELOW:

1- a) FINDS A FLOW PATTERN THAT SATISFIES A GIVEN TERMINAL REQUIREMENT AT MINIMUM COST OR

b) FINDS THE FLOW FROM I TO J THAT GIVES A MAXIMUM FLOW AT MINIMUM COST.

2- FINDS THE NEW ARC CAPACITIES THAT MUST BE ADDED TO THE NETWORK TO:

a) ACHIEVE A GIVEN REQUIRED INCREASE IN FLOW FROM I TO J OR

b) STAY WITHIN A GIVEN BUDGET AS THE NETWORK IS EXPANDED.

INPUT:

NO. OF NODES
4

NO. OF ARCS
5

NODE TABLE

NODE NO.	LABEL
1	TORONTO
2	OTTAWA
3	MONTREAL
4	QUEBEC

ARC TABLE

FROM NODE	TO NODE	MIN. FLOW	MAX. FLOW	RENTAL COST PER UNIT FLOW	EXPANSION COST PER UNIT CAPACITY	INITIAL FLOW
1	2	5	15	10	50	0
1	3	0	25	12	40	0
2	3	5	15	20	15	0
2	4	5	10	20	100	0
3	4	0	30	10	40	0

PLANNED INVESTMENT FOR EXPANSION	500

PROCESSING:

MAIN PROGRAM

PROGRAM NETPLAN (,K)

OUTPUT:

OPTIMAL FLOW PATTERN REPORT

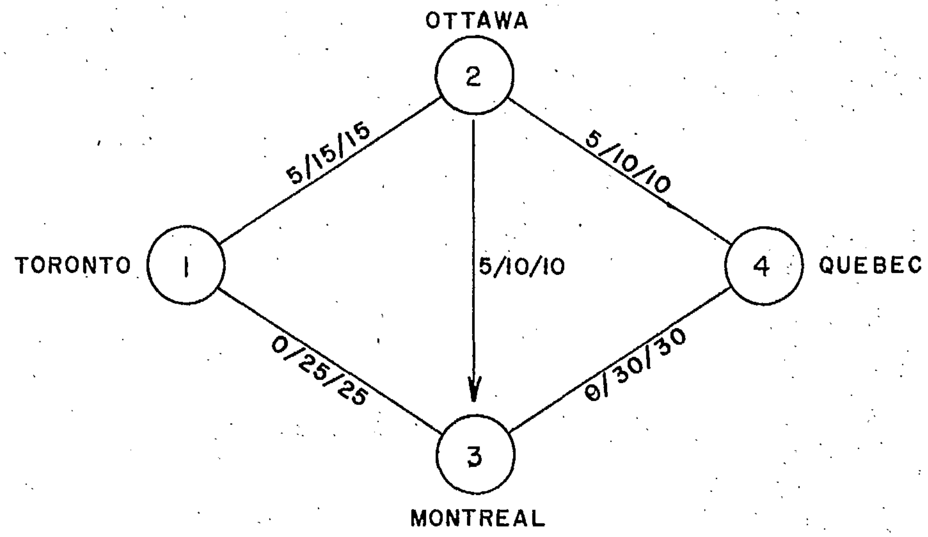
FROM	TO	MIN. FLOW	MAX. FLOW	CURRNT FLOW	RENTAL COST
TORONTO	MONTREAL	0	25	25	12
TORONTO	OTTAWA	5	15	15	10
OTTAWA	QUEBEC	5	10	10	20
OTTAWA	MONTREAL	5	15	5	20
MONTREAL	QUEBEC	0	30	30	10
TOTAL TRANSP. COST FROM TOR. TO QUE.					1,050
CURRENT FLOW FROM TORONTO TO QUEBEC					40

ARC CAPACITY EXPANSION REPORT

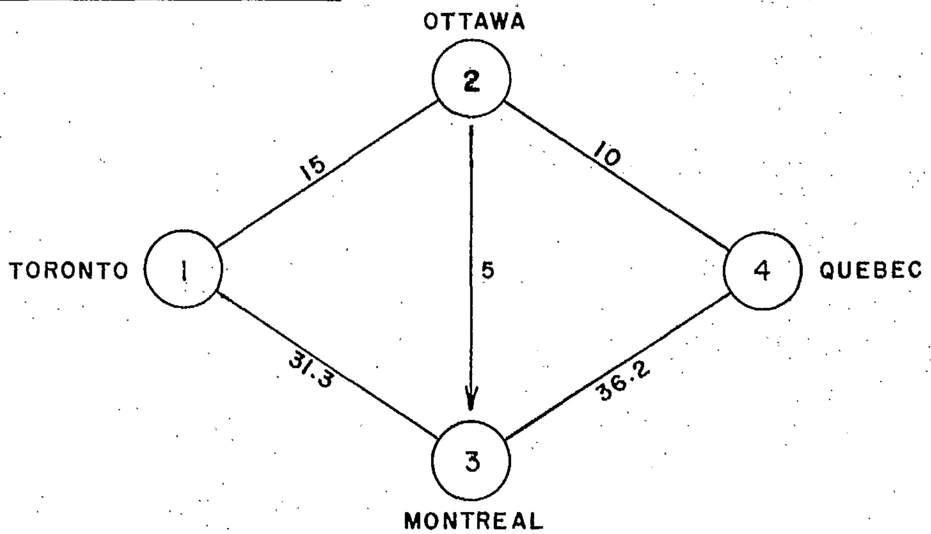
FROM	TO	OLD MAX.	NEW MAX.	CAP. CHANGE	EXP'N COST	UNIT ARC COST
TORONTO	MONTREAL	25	31.3	6.2	40	250.0
TORONTO	OTTAWA	15	15.0	0	50	0
OTTAWA	QUEBEC	10	10.0	0	100	0
OTTAWA	MONTREAL	15	5.0	0	15	0
MONTREAL	QUEBEC	30	36.2	6.2	40	250.0
TOTAL INVESTMENT FOR EXPANSION						500
TOTAL FLOW TOR. TO QUE. AFTER EXPANSION						46.2

RESULTANT FLOW PATTERNS

A- FROM OPTIMAL FLOW REPORT



B- FROM CAPALITY EXPANSION REPORT



RUNNING PROGRAMS
- SHORT1(,K)
- SHORT2(,K)
- NETPLAN(,K)
- NETSYM1(,K)

STEP 1- DIAL-UP THE SYSTEM:

One of the following telephone ports may be dialed: 828-2754; 996-7051, EXT. 505 to 508; 996-6723. The teletype should be turned on and then the number should be dialed. If a high pitched tone is observed, the "dial-up" has been successful and the telephone receiver may be inserted into the modem.

STEP 2- LOG ONTO THE SYSTEM:

The system will respond to the "dial-up" with those characters underlined below. The user should type in all other characters

```
BTM  SYSTEM  IS  UP
16/9/71       14:30
! LOGIN:  PLANNING,1004S,POLICY
  ID=5
```

STEP 3- LOAD THE PRØGRAM AND INITIATE EXECUTIØN:

```
! LOAD
  ELEMENT FILES:  SHØRT1B,SHØRT2B,NETPLANB or NETSYM1B
  OPTIONS
  F:1
  F:
  SEVERITY LEVEL=0
  XEQ? Y
```

STEP 4- INPUT DATA AS REQUESTED:

The program goes into conversational mode, asking for the data as required and terminating on job completion.

STEP 5- LOG OFF THE SYSTEM:

```
! BYE
16/9/71      14:35
```

SUMMARY OF PROGRAMMING PACKAGES
AVAILABLE FROM
TERRESTRIAL PLANNING BRANCH

PROGRAM NAME	PROGRAM DESCRIPTION
SHORT 1(,K)	THIS PROGRAM'S SYNTHESIZES FROM GIVEN TERMINAL REQUIREMENTS AND ARC COSTS, A COMMUNICATIONS NETWORK IN WHICH COMMUNICATION BETWEEN ALL PAIRS OF NODES EXISTS AT THE SAME TIME (SIMULTANEOUS TRANSMISSION). TOTAL NETWORK COST IS MINIMIZED.
SHORT 2(,K)	SAME AS ABOVE EXCEPT THAT ONLY ONE PAIR OF TERMINALS COMMUNICATES AT ONE TIME (TIME-SHARED COMMUNICATIONS). TOTAL NETWORK COST IS REDUCED.
NETPLAN(,K)	GIVEN A COMMUNICATIONS NETWORK WITH ARC CAPABILITIES, ARC RENTAL COSTS AND ARC EXPANSION COSTS ALSO GIVEN FOR A GIVEN PAIR OF NODES, THIS PROGRAM COMPUTES THE OPTIMAL NEW FLOW PATTERNS AND THE OPTIMAL EXPANSION PATTERNS.
NETSYM 1(,K)	GIVEN THE TERMINAL REQUIREMENTS AND THE ARC CONSTRAINTS A TIME-SHARED COMMUNICATIONS, THIS PROGRAM SYNTHESIZES A NETWORK IN WHICH ALL TERMINAL REQUIREMENTS ARE EXACTLY SATISFIED.
MAT(,PROB)	THIS PROGRAM HAS A NUMBER OF VARIOUS MATRIX OPERATIONS, SUCH AS MATRIX INVERSION, FUNCTIONS OF A MATRIX, ETC. IT ALSO PERFORMS VARIOUS SYSTEM RELIABILITY SIMULATIONS.
MAX(,MIN)	THIS PROGRAM ALLOWS OPTIMAL STRATERGIES TO BE OBTAINED WITH CORRESPONDING MAXIMUM GAIN OR MINIMUM LOSS, FOR VARIOUS DECISION MAKING PROBLEMS.