June, 1976 Report ESL-R-667

DYNAMIC FILE ALLOCATION IN A COMPUTER NETWORK

by

Francisco de Asís Ros Perán

This report is based on the unaltered thesis of Francisco de Asís Ros Perán, submitted in partial fulfillment of the requirements for the degree of Master of Science at the Massachusetts Institute of Technology in June, 1976. This research was conducted at the Decision and Control Sciences Group of the M.I.T. Electronic Systems Laboratory with partial support provided under ARPA Contract N00014-76-C-1183.

Electronic Systems Laboratory
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

DYNAMIC FILE ALLOCATION IN A COMPUTER NETWORK

by

Francisco de Asís Ros Perán

Ingeniero de Telecomunicación

Universidad Politécnica de Madrid

1972

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May, 1976

Francisco

Signature of Author...
Department of Electrical Engineering
and Computer Science, May 7, 1976

Adrian Segall

Certified by..
Thesis Supervisor

Accepted by...
Chairman, Departmental Committee on Graduate
Students

DYNAMIC FILE ALLOCATION IN A COMPUTER NETWORK

by

Francisco de Asís Ros Perán

Submitted to the Department of Electrical Engineering and Computer Sciences on May 7, 1976 in partial fulfillment of the requirements for the Degree of Master of Science.

ABSTRACT

One of the main reasons computer networks are a major area of great attention and development today is their capability to provide the facilities for common use of data bases and information files by all computers in the system.

When a file is used by several computers in the network, it can be stored in the memory of at least one of them and can be accessed by the other computers via the communication channels. In general the cost or querying is reduced as we increase the number of copies in the system. On the other hand, storage costs, limitations on the size of the memories and the cost of updating (every copy must be updated) will dictate decreasing of the number of copies. Furthermore if the parameters of the system are time-varying, or if the exact pattern of the rates of demand is unknown or some non negligible possibility of node or link failures is expected, then some kind of dynamic approach must be used.

This thesis considers the problem of optimal dynamic file allocation when more than one copy is allowed to exist in the system at any given time. A general model to handle this problem including updating traffic and the possibility of node failures will be developed. The evolution of the system is represented as a finite state Markov process and Dynamic programming will be used for the solution of the optimization problem.

The use of two types of control variables, one for adding new copies to the system and the other for erasing copies, gives the model certain properties that permit the construction of an efficient algorithm to solve the optimization problem. Within the framework of the developed model the addition of the updating traffic and the possibility of node failures present no important difficulties. Furthermore the model can easily handle the problem of constraints in the maximum or minimum number of copies. In the last chapter the model and algorithms are applied to several numerical examples.

Thesis Supervisor: Adrian Segall

Title: Assistant Professor of Electrical Engineering
and Computer Science

ACKNOWLEDGEMENTS

Thanks are due to Prof. Segall who was the original source of this thesis topic and many of the ideas here exposed. Rafael Andreu was very helpful in the computer programming part of the work due to his considerable experience in this area. Ramón Bueno contributed with his bilingual ability to the final English structure of some parts of the text. And finally Camille Flores' typing skills permitted to meet the deadline.

This work has been supported by a fellowship from the Fundación del Instituto Tecnológico para Postgraduados, Madrid (Spain) and by the Advanced Research Project Agency of the Department of Defense (monitored by ONR) under contract NOOO14-75-C1183.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

I-1 General Setting and Description of the Problem

"The time sharing industry dominated the sixties and it appears that computer networks will play a similar role in the seventies. The need has now arisen for many of these time-shared systems to share each others' resources by coupling them together over a communication network thereby creating a computer network" (L. Kleinrock[1]).

We define a computer network to be an interconnected group of independent computer systems communicating with each other and sharing resources such as programs, data, hardware, and software.

The increasing interest in this area is the cause for a continuously growing number of articles, books and projects related to computer networks [2] - [7], [24]. The reasons why these types of networks are attractive are widely exposed throughout the literature in this field.

a) sharing of data base, hardware resources, program and load

b) remote data processing

c) access to specialized resources

d) recovery of information from a remote node in case of node failure

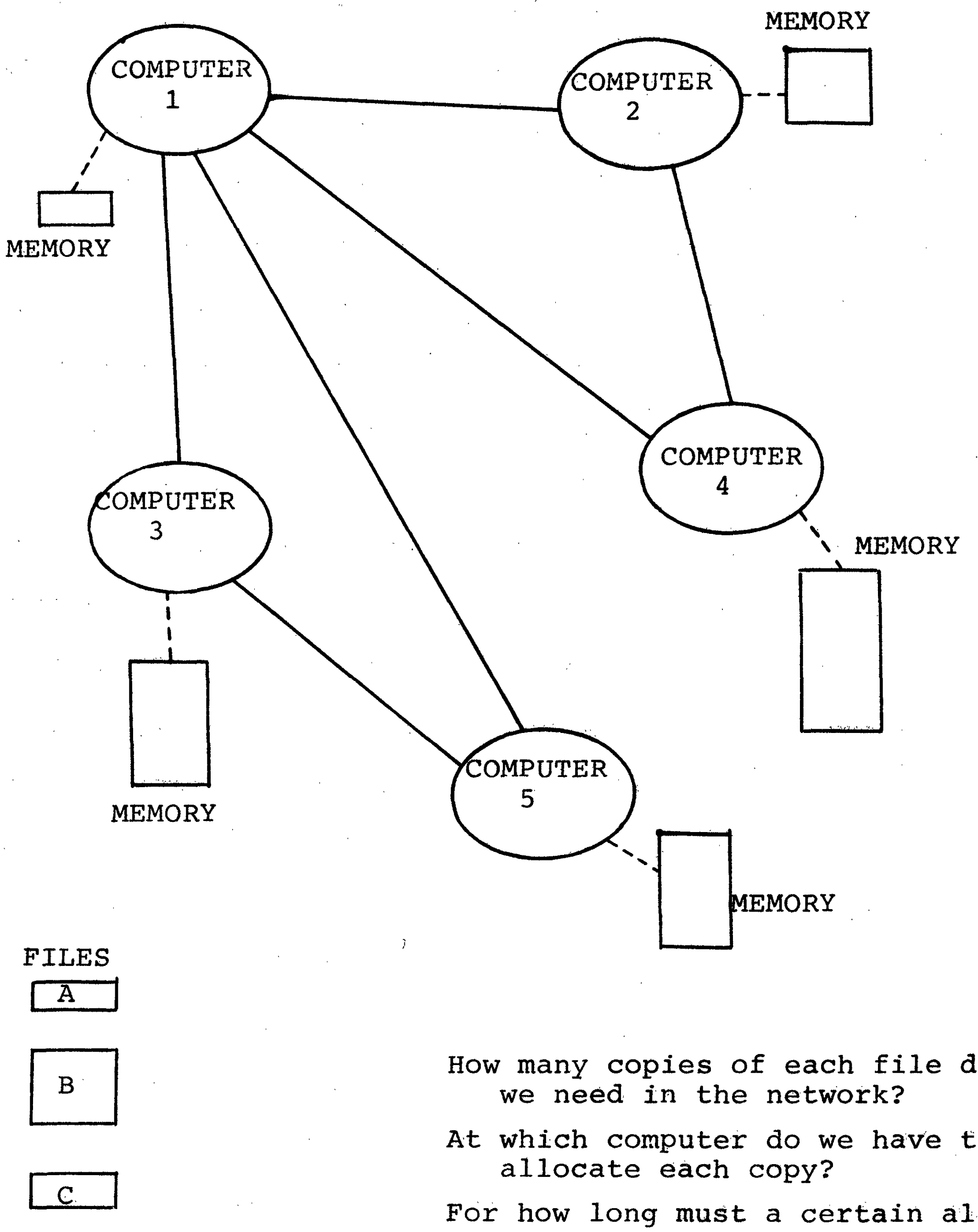
e) decentralization of operations and need to transfer information from one point to another etc.

One of the main reasons computer networks are a major area of great attention and development today is their capability to provide the facilities for common use of data bases and information files by all computers in the system. This work deals with the problem of the information allocation to be shared by the computers in the network. Such a network is displayed in Fig. 1

When a file is used by several computers in the network, it can be stored in the memory of (at least) one of them and can be accessed by the other computers via the communication channels. In general, the cost of querying is reduced as we increase the number of copies in the system. On the other hand, storage costs, limitations on the size of the memories and the cost of updating (every copy must be updated) will dictate decreasing of the number of copies.

The problem of how many copies of the files are to be kept and their allocation is the main subject of this Thesis.

Most of the previous work in the area of file allocation has been devoted to the analysis of the problem under static approximations, that is, assuming that all parameters of the system are known a priori and basing the design on their average value over the period of operation of the system. The location of the files is then considered fixed



How many copies of each file do we need in the network?

At which computer do we have to allocate each copy?

For how long must a certain allocation distribution remain unchangeable?

etc.

Fig. I.1. General representation of a computer network reflecting the problem of file allocation

for the whole operating period.

An early work in this field was a paper by Chu [8]. The criterion of optimality used in [8], is minimal overall operating costs. The model considers storage and transmission costs, request and updating of the files and a limit on the storage capacity of each computer. The model searches for the minimum of a non-linear zero-one cost equation which can be reduced to a linear zero-one programming problem.

Another work is a paper by Casey [9]. He considers a mathematical model of an information network of n nodes, some of which contain copies of a given data file. Using a simple linear cost model for the network, several properties of the optimal assignment of copies of the file are demonstrated. One set of results expresses bounds on the number of copies of the file that should be included in the network, as a function of the relative volume of query and update traffic. The paper also derives a test useful in determining the optimum configuration.

Of very recent appearance is a paper by Mahmoud and Riordon [22]. In this paper the problems of file allocation and capacity assignment in a fixed topology distributed computer network are simultaneously examined. The objective, in that analysis, is to allocate copies of information files to network nodes and capacities to network links so that a minimum cost is achieved subject to network delay and file availability constraints. The deterministic solution for a medium size problem is intractable due to the large amount of computation so that an heuristic algorithm is proposed.

A quite different analysis in which the important quantity to be optimized is the service time, instead of the operating cost, is done by Belokrinitskaya et all. [10]. The analysis results in a zero-one nonlinear programming problem (that can be linearized), similar to the one in [8].

In the above mentioned works, the problem is considered under static conditions and using average values of the parameters.

If the parameters of the system are time-varying, however, or if the exact pattern of the rates of demand is unknown or some non negligible possibility of node or link failures is expected, then some kind of dynamic approach must be used.

It has been only recently that the first studies of these problems, from the dynamic point of view, have begun to appear. In a work by A. Segall [11] the problem of finding optimal policies for dynamical allocation of files in a computer network that works under time-varying operating conditions is studied. The problem is considered under the assumption that the system keeps one copy of each file at any given time. The case when the rates of demand are not perfectly known in advance is also treated. Only a prior distribution and a statistical behaviour are assumed, and the rates have to be estimated on-line from the incoming requests.

The problem of optimally allocating limited resources, among competing processes, using a dynamic programming approach is studied in [12]. A dynamic programming approach is also suggested for the problem of minimizing the costs of data storage and accesses in [25]. Here two different types of accessing costs are considered. The accessing cost will depend on whether a record is to be read or to be written (migration). A different approach to the same problem is taken in [26]. A two-node network with unknown access probabilities is considered. The problem is to set up a sequential test which determines the earliest moment at which migration leads to a lower expected cost.

The present work considers the problem of optimal dynamic file allocation when more than one copy are allowed to exist in the system at any given time. A general model to handle this problem including updating traffic and the possibility of node failures will be developed.

The evolution of the system is represented as a finite-state Markov process and dynamic programming will be used for the solution of the optimization problem.

I.2. Summary of Results

A model for the analysis of optimal dynamic file allocation is introduced. The use of two types of control variables, one for adding new copies to the system and the other for erasing copies, gives to the model certain properties that permit the construction of an efficient and relatively simple algorithm to solve the optimization problem. Among others, the algorithm is efficient due to the fact that it computes only the nonzero transition probabilities. A detailed set of flow-charts and Fortran program listings are given for all the operations and calculations that take place in the optimization process.

Within the same framework the incorporation of node failures presents no important difficulties, except for increasing the number of states. Some kind of constraints in the state space, those that could be represented as reductions in the set of admissible states, are also easily handled by the model.

In the last chapter we apply the algorithms to several numerical examples. For the case of constant rates of demand with no failures in the computers the corresponding Markov processes have a trapping state. For these processes it will be shown that the general dynamic programming algorithm need not be implemented, and a much quicker answer to the optimization process can be found.

For the more general case of constant parameters with possibility of node failures included, quick convergence to the steady state optimal dynamic decision policy was found for all examples.

Finally it will be shown that having a completely simmetric network (equal parameter values for all computers and links) will allow a considerable reduction in the number of states.

A more detailed exposure of results can also be found in the chapter dedicated to "Conclusions and Open Questions".

I.3. Chapter Outline

In chapter II we begin with the description of the model. We first state the general hypothesis and basic assumptions to be considered throughout the study and continue with the description of the operation procedure. We indicate the objective function and define the control and allocation variables. The chapter ends with the definition of the state and the description of the dynamic equations of the system.

In chapter III Stochastic dynamic programming is applied to the model to determine the optimal allocation strategy. First we will write the recursive equations for a simple network with only two computers and then we will see how easily these equations generalize to any number of computers. We finish the chapter indicating how the model can handle the problem of certain constraints in the state space.

In chapter IV we present the problem in its more general framework with the inclusion of the updating traffic and the possibility of node failures. As in chapter III, we first write the recursive equations for a network with two computers and then generalize them to any number of computers. At this point we give a very detailed set of flow-charts, showing how to compute the different matrices and vectors of the recursive equations and how to carry out the whole optimization process.

Chapter V deals with numerical applications. Using the insight gained from numerical answers some additional analytical results are developed.

A few pages dedicated to general conclusions and further work to be done in this area will follow this chapter.

Two appendices, A and B, expanding results of chapters II and III will also be added. A third appendix contains a set of Fortran program listings corresponding to the most significant flow-charts of previous chapters. These programs have been used to implement the numerical applications of chapters V. Auxiliary subroutines are also listed.

CHAPTER II

DESCRIPTION OF THE MODEL

II.1 Characteristics, Basic Assumptions and Operation Procedure

We shall make here several simplifying assumptions, that are still consistent however with the models appearing in real networks. We shall assume that the files are requested by the computers according to mutually independent processes (with statistics to be specified presently) and also that the files are sufficiently short. Moreover the communication lines are taken to have sufficient capacity and the computers sufficient memory, so that the transmission of the file takes a very short time and there is no restriction on how many files a computer can carry. Under these assumptions, it is clear that in fact the files do not interfere with each other, and we can therefore treat each file separately.

The analysis will be done in discrete time, assuming the existence of a central synchronizing clock. It will be considered that with previous assumptions the time interval between clock impulses is long enough to allow the execution of all the necessary operations to take place in it (request arrivals, "reading" of present state, implementation of optimal decisions, etc.).

In this chapter the possibility of node failures will not be included in the model. This extension, together with the inclusion of the updating traffic, will be left to Chapter IV.

We may summarize the assumptions as follows:

1) No failures in the network (relaxed in Chapter IV)
2) Channels with sufficient capacity (or sufficiently short files)
3) Sufficiently large storage capacity at each computer
4) Requesting according to mutually independent processes
5) Files are treated separately (according to former assumptions the files do not interfere with each other)
6) The analysis is done in discrete time

The proposed procedure is similar to that proposed in reference [11], with the only difference being that we now allow more than one copy at each instant of time (the way the updating traffic is taken in consideration will be described in Chapter IV).

The procedure is illustrated in Fig. II.1 and can be described as follows: Suppose a certain number of copies is stored at time t in the memories of a set of computers, say I. If at time t the file is requested only by computers in the set I then no transmission cost is incurred and a decision has to be made, whether to erase some of the copies from I (with the specification of the particular copies to

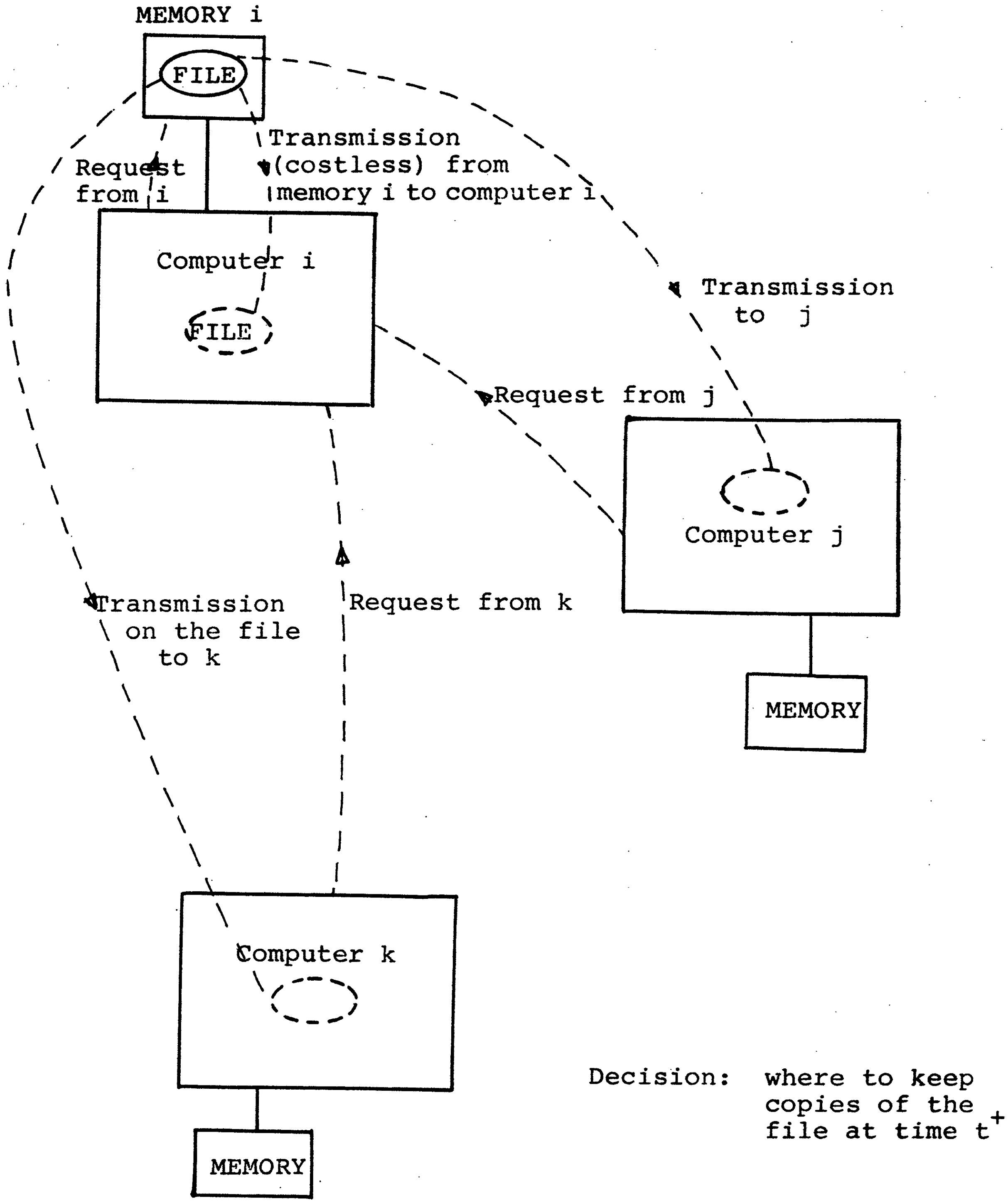

MEMORY i
FILE
Request from i
Transmission (costless) from memory i to computer i
Computer i
FILE
Transmission to j
Request from j
Computer j
MEMORY
Transmission on the file to k
Request from k
Computer k
MEMORY
Decision: where to keep copies of the file at time t+

Fig. II.1. Illustration of the Operation Procedure

be erased) or to keep the same number of copies. If, on the other hand, the file is also requested by other computers not in I, then the file is transmitted for use to these computers and a new collection of copies, say J, appears in the system at time t^+. A similar decision now has to be made but with the set J instead of the set I

The restriction of reallocating the file only in conjunction with a regular transmission is reasonable for this model, because if a change of location is decided upon, one might as well wait until the file is requested for the next time by the appropriate computer, otherwise it is conceivable that the file might be transferred back and forth, without anybody actually using it.

II.2 Data, Parameters and Variables

In this section part of the notation used in the study will be introduced.

Consider a completely connected network of NC computers. The requests of the file by the computer will be modeled as mutually independent Bernoulli processes with rates $\theta_i(t)$, $i = 1,..NC$, that is

$$P_r\{n_i(t) = 1\} = 1 - P_r\,\{n_i(t) = 0\} = \theta_i(t) \qquad (2.1)$$

$$i = 1,..NC$$

where $n_i(t) = 1$ indicates that the file has been requested by computer i at time t. The rates $\Theta_i(t)$ are assumed to be known for all computers and instants of time.

We define the variables

$$y_i(t) = \begin{cases} 1 & \text{if there is a copy stored at computer } i \text{ at time } t \\ 0 & \text{Otherwise} \end{cases} \qquad (2.2)$$

$$i = 1,..NC$$

The condition of having at least one copy of the file in the system at any instant of time can be analytically expressed as

$$\sum_{i=1}^{NC} y_i(t) \geq 1 \qquad \forall t \in \left[0,T\right] \qquad (2.3)$$

where T is the whole period of operation.

The operation costs are

C_i = storage cost per unit time per copy at memory i

C_{ij} = communication cost per transmission from computer i to computer j

$$i, j = 1,... NC \qquad i \neq j$$

We will assume $C_{ii} = 0 \quad \forall i$

It is assumed that these costs are time-invariant; the case with time-varying costs can be handled by simply writing $C_i(t)$ and $C_{ij}(t)$ throughout the paper.

II.3 Objective Function

Supposing that the user accesses that copy of the file that minimizes his communication cost and, denoting symbolically by I(t) the set of nodes having a copy at time t, we can write the expression for the total expected cost over the period $[0,T]$ as

$$C = E\sum_{t=0}^{T}\left[\sum_{i=1}^{NC} c_i y_i(t) + \sum_{i=1}^{NC} (1-y_i(t))n_i(t)\min_{k\epsilon I(t)} c_{ki}\right] \quad (2.4)$$

The first sum in the bracket represents the total storage cost at time t and the second sum is the total transmission cost. We can see that summands contributing to the transmission cost are those with $y_i(t) = 0$ and $n_i(t) = 1$ only, that is, those coming from computers that do not have the file and have had a request.

The goal is to design a closed-loop control that will dynamically assign the location of the file and will minimize the defined expected cost. We introduce the control variables in the next section.

II.4 Control Variables and Restatement of the Objective Function

We will define two types of control variables. One will correspond to the erasure process and the other one to the writing process. The separation of these two operations in in two types of control variables will simplify significantly the amount of notation.

The variables are

$$\varepsilon_i(t) = \begin{cases} 1 \text{ — } & \left\{ \begin{array}{l} \text{if the decision is to erase the copy from } i \\ \text{at time } t^+ \text{ assuming the copy was there at} \\ \text{at time } t \text{ (i.e. } y_i(t) = 1) \end{array} \right. \\ 0 \text{ — } & \text{otherwise} \end{cases} \qquad (2.5)$$

$$\alpha_i(t) = \begin{cases} 1 \text{ — } & \left\{ \begin{array}{l} \text{if the decision is to keep a copy in } i \text{ at} \\ \text{time } t^+ \text{ assuming that the copy was not there} \\ (y_i(t)=0) \text{ and there was a request from that} \\ \text{computer } (n_i(t) = 1) \end{array} \right. \\ 0 \text{ — } & \text{otherwise} \end{cases} \qquad (2.6)$$

$$i = 1,..NC$$

These definitions require the introduction of the concept of active control variables. It will be said that the variable $\varepsilon_i(t)$ is active if $y_i(t) = 1$ and that $\alpha_i(t)$ is active if $y_i(t) = 0$. Due to these definitions $\alpha_i(t)$ and $\varepsilon_i(t)$ cannot be simultaneously active. From definitions (2.5) and (2.6) the nonactive variables will always be equal zero. Therefore only active variables will be considered throughout the analysis.

With the previous notations, the dynamic evolution of the system is:

a) $$y_i(t+1) = y_i(t)\left[1-\varepsilon_i(t)\right] + \left[1-y_i(t)\right]\alpha_i(t)n_i(t)$$

$$i = 1,1,..NC \qquad (2.10a)$$

$$\text{iff} \sum_{i=1}^{NC} (\text{right hand side}) \neq 0$$

b) $$y_i(t+1) = y_i(t) \qquad i = 1,2,..NC \qquad (2.10b)$$

$$\text{iff} \sum_{i=1}^{NC} (\text{right hand side of (2.10a)}) = 0$$

Equation (2.10b) shows that if our decision variables are such that all copies of the file will be erased, then no decision variable is actually implemented, and therefore the system remains in the previous state. Otherwise, the system evolves according to equation (2.10a) namely computer i will have a copy at time (t+1) if

i) it had a copy at time t ($y_i(t) = 1$) and the decision was not to erase it ($\varepsilon_i(t) = 0$) <u>or</u>

ii) it did not have a copy at time t and there was a request from computer i ($n_i(t) = 1$) and a decision to write the file into memory i was taken ($\alpha_i(t) = 1$).

The optimization problem could then be stated as follows: Given the dynamics (2.10), find the optimal control policies

$$\left.\begin{array}{l}\varepsilon_i^*(t)\\ \\ \alpha_i^*(t)\end{array}\right\} \begin{array}{l} i = 1,1,..,NC \\ \\ t = 1,2,..T\end{array}$$

and the initial locations $y_i(0)$, $\forall i$, so as to minimize the expected cost (2.4).

Hence we have a dynamic system in which the inputs are a sequence of decisions made at various stages of the evolution of the process, with the purpose of minimizing a cost. These processes are sometimes called multi-stage decision processes [15].

II.5 Definition of State and Dynamics of the System

Being at a certain instant of time, in the optimization process, the only information needed, given the fact that the request rates are perfectly known, is the identification of the computers that have a copy of the file at that time. With only this information we can continue the optimization process and the past is inmaterial as far as the future is concerned. Therefore the location of the copies at any instant of time summarizes the information needed at that instant (together with the rates) and the problem then is to find an optimal policy for the remaining stages in time.

The state of the system will be defined, at time t, as the location of the copies of the file at that time and it will be represented by a vector with NC binary components, having a zero in the places corresponding to computers that do not have the file and a one in the places of computers having a copy. These vectors will be named by the decimal number whose binary representation is the NC- dimensional vector and will be represented by a capital Y.

Therefore the state at time t will be the column vector

$$Y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \cdot \\ \cdot \\ \cdot \\ y_{NC}(t) \end{bmatrix} \qquad (2.11)$$

or alternatively the state of the system at time t is m(t) where

$m(t)$ = decimal number with binary representation given by the sequence $y_1(t)\ y_2(t)$--$y_{NC}(t)$

$m = 1,2,\ldots,M$

$$M = 2^{NC} - 1$$

$m = 0$ will not be a valid state because it corresponds to the case of having no copies in the system and this situation has to be avoided. Thus the previously stated condition

$$\sum_{i=1}^{NC} y_i(t) \geq 1$$

is translated to this notation as $m \neq 0$

The dynamics of the state are easily obtained from the dynamics of the allocation variables: we only have to substitute for each component of the vector defining the state

$$\begin{bmatrix} y_1(t+1) \\ y_2(t+1) \\ \cdot \\ \cdot \\ \cdot \\ y_{NC}(t+1) \end{bmatrix} = \begin{bmatrix} 1-\varepsilon_1(t) & & \cdot & \cdot & \cdot & 0 \\ 0 & 1-\varepsilon_2(t) & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & \cdot & 1-\varepsilon_{NC}(t) \end{bmatrix} \begin{bmatrix} y_1(t) \\ y_2(t) \\ \cdot \\ \cdot \\ \cdot \\ y_{NC}(t) \end{bmatrix} +$$

$$+ \begin{bmatrix} \alpha_1(t)n_1(t) & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & \alpha_2(t)n_2(t) & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \\ \cdot & 0 & \cdot & \cdot & \cdot & \\ 0 & & & & & \alpha_{NC}(t)n_{NC}(t) \end{bmatrix} \begin{bmatrix} 1-y_1(t) \\ 1-y_2(t) \\ \cdot \\ \cdot \\ \cdot \\ 1-y_{NC}(t) \end{bmatrix}$$

(2.12a)

iff right hand side of (2.12a) $\neq \underline{0}$ and

$Y(t+1) = Y(t)$ iff right hand side of (2.12a) $= \underline{0}$ (2.12b)

To clarify further these ideas let us write the ordered sequence of events that take place at any time t

- at time t^- the optimal $\varepsilon_i^*(t)$ and $\alpha_i^*(t)$ are computed
- at time t the requests arrive
- at time t^+ the optimal decisions are activated if in doing so the system does not go to state 0 (that is, if not all existing copies are erased). Otherwise the system does not change state.

This sequence of events is illustrated in Fig. II-2.

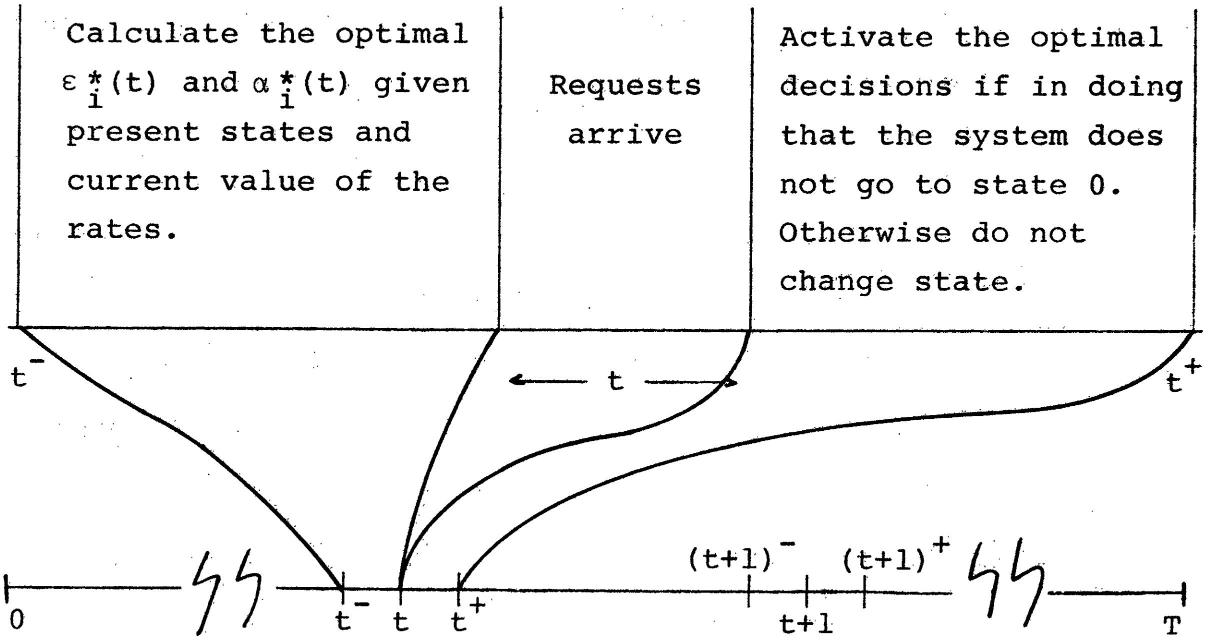

Calculate the optimal $\varepsilon_i^*(t)$ and $\alpha_i^*(t)$ given present states and current value of the rates.
Requests arrive
Activate the optimal decisions if in doing that the system does not go to state 0. Otherwise do not change state.
t^-
t
t^+
0
t^-
t
t^+
$(t+1)^-$
$(t+1)^+$
t+1
T

Fig. II-2. Sequence of events at any time t

II.6 Some Useful Properties of the Model

So far the main structure of the model has been described. In this section, we describe some of the properties of the model. First of all we will look at the transitions among states.

Recall from section II.4 that the active variables are defined as

$$\varepsilon_i(t) \text{ is active if } y_i(t) = 1$$

$$\alpha_i(t) \text{ is active if } y_i(t) = 0$$

Hence these variables are uniquely determined by the state. For instance, having a network with five computers (NC=5) and being in state eleven (01011) the active variables are

$$\begin{array}{ccccc} \underline{\text{State}} & & \underline{y_1\ y_2\ y_3\ y_4\ y_5} & & \underline{\text{active variables}} \\ Y = 11 & \longrightarrow & 0\quad 1\quad 0\quad 1\quad 1 & \longrightarrow & \alpha_1\ \varepsilon_2\ \alpha_3\ \varepsilon_4\ \varepsilon_5 \end{array}$$

withα's corresponding to places where there is a 0 (no file in the memory of that computer) and ε's to places where there is a 1 (there is a copy at that computer). The non-active variables will then beε_1, α_2, ε_3, α_4 and α_5 and we saw, also in section II.4, that their value is equal zero no matter which decision is taken, so we can omit them.

Suppose now that the optimal decision at a given time is:

- erase copies from computers 2 and 5
- keep a copy at computer 3

or in terms of the control variables

$$u = (\alpha_1\ \varepsilon_2\ \alpha_3\ \varepsilon_4\ \varepsilon_5) = (0\ \ 1\ \ 1\ \ 0\ \ 1)$$

Thus, if there is a request from computer 3, the system will go to state

$$\begin{array}{ccccc} y_1 & y_2 & y_3 & y_4 & y_5 \\ \hline 0 & 0 & 1 & 1 & 0 \end{array} \quad \longrightarrow \quad \text{state } 6$$

and if there is no request from computer 3 the system will go to

$$\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 1 & 0 \end{array} \quad \longrightarrow \quad \text{state } 2$$

Because of the unique correspondence in the notation we see that it is equivalent to say that the decision is

$$u = (\alpha_1\ \varepsilon_2\ \alpha_3\ \varepsilon_4\ \varepsilon_5) = (0\ \ 1\ \ 1\ \ 0\ \ 1)$$

or

"go to state 6"

For the sake of simplicity these two forms will be interchangeably used.

From the above analysis it can be seen that

(initial state) $\oplus$ (decision vector) = (final desired state)

(decision vector) = (initial state) $\oplus$ (final desired state)

where $\oplus$ means "exclusive or". This is so because if a control variable has a value 1 we have to change the value of the allocation variable in the transition, while if the value is 0 there is no change in the transition. This kind of operation is exactly the "exclusive or" addition. In particular for the example above

$$\begin{array}{lcl} 11 \longrightarrow & & 0\ \ 1\ \ 0\ \ 1\ \ 1 \\ & + & \\ \alpha_1\ \varepsilon_2\ \alpha_3\ \varepsilon_4\ \varepsilon_5 \longrightarrow & & 0\ \ 1\ \ 1\ \ 0\ \ 1 \\ \hline & & 0\ \ 0\ \ 1\ \ 1\ \ 0 \longrightarrow 6 \end{array}$$

This property will permit to easily write a tableau showing the transitions among states assuming the requests arrive from all computers.

The transition tableau for the case NC=2 is shown in Fig. II-3 and for NC=3 in Fig. II-4.

For simplicity, the place corresponding to non-active variables are left empty.

$\underline{Y}(t)$	Active variables $\underline{Y}(t+1)$	1=0 1	2=1 0	3=1 1
1=0 1	ε_2 α_1	0 0	1 1	0 1
2=1 0	ε_1 α_2	1 1	0 0	0 1
3=1 1	ε_1 ε_2	1 0	0 1	0 0

Fig. II-3. Transition tableau, for the case NC=2

(assuming deterministic transitions)

Y(t)	Y(t+1) Act. Variables	1=0 0 1	2=0 1 0	3=0 1 1	4=1 0 0	5=1 0 1	6=1 1 0	7=1 1 1
1=0 0 1	ε_3 α_1 α_2	0 0 0	1 0 1	0 0 1	1 1 0	0 1 0	1 1 1	0 1 1
2=0 1 0	ε_2 α_1 α_3	1 0 1	0 0 0	0 0 1	1 1 0	1 1 1	0 1 0	0 1 1
3=0 1 1	ε_2 ε_3 α_1	1 0 0	0 1 0	0 0 0	1 1 1	1 0 1	0 1 1	0 0 1
4=1 0 0	ε_1 α_2 α_3	1 0 1	1 1 0	1 1 1	0 0 0	0 0 1	0 1 0	0 1 1
5=1 0 1	ε_1 ε_3 α_2	1 0 0	1 1 1	1 1 0	0 1 0	0 0 0	0 1 1	0 0 1
6=1 1 0	ε_1 ε_2 α_3	1 1 1	1 0 0	1 0 1	0 1 0	0 1 1	0 0 0	0 0 1
7=1 1 1	ε_1 ε_2 ε_3	1 1 0	1 0 1	1 0 0	0 1 1	0 1 0	0 0 1	0 0 0

Fig. II-4. Transition tableau for the case NC=3 (assuming deterministic transitions).

These tableaus will prove later to be of great utility for the construction of the transition probabilities among states. As we said before, we can write this tableau mechanically and this is important in computer calculations.

Summarizing, for a network with NC computers, the steps are the following:

1 - If the system is in state m, write m in base 2 with NC digits.

2 - Assign a control variable α to the places where the digit is 0 and a variable ε to the places where the digit is 1.

3 - To obtain the value of these control variables in a transition from m to n, compute $m \oplus n$, where n is also written in base 2 with NC digits, and assign the values of the resulting digits to the corresponding control variables.

4 - To obtain the mth row of the tableau repeat step 3 for values of n from 1 to M $(M = 2^{NC}-1)$

5 - To obtain all the rows of the tableau repeat from step 1 for values of m from 1 to M.

The flow-chart corresponding to these five steps is shown in fig. II-5.

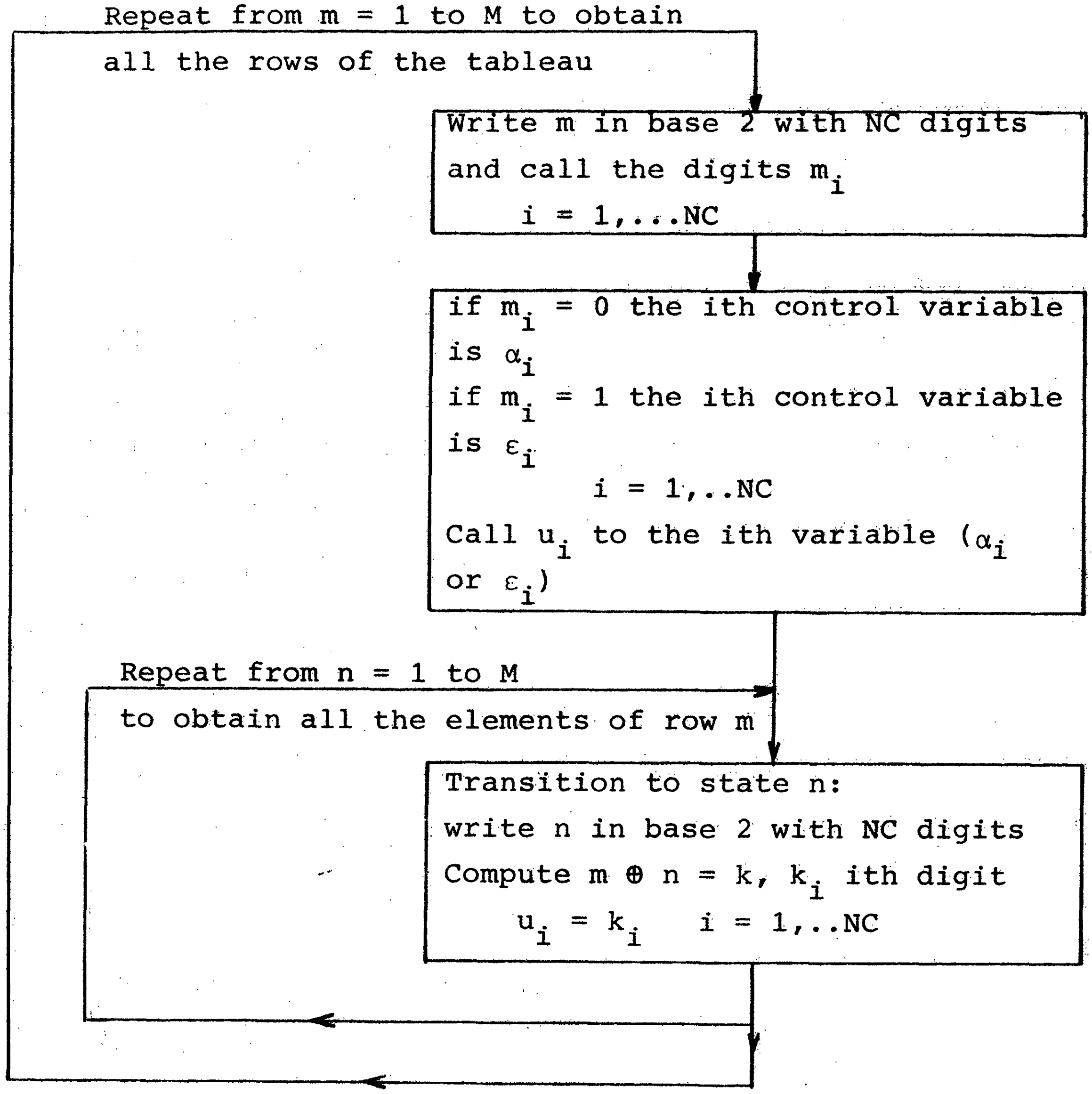

Repeat from m = 1 to M to obtain all the rows of the tableau
Write m in base 2 with NC digits and call the digits m_i $i = 1,\ldots NC$
if $m_i = 0$ the ith control variable is α_i
if $m_i = 1$ the ith control variable is ε_i
$i = 1,..NC$
Call u_i to the ith variable (α_i or ε_i)
Repeat from n = 1 to M to obtain all the elements of row m
Transition to state n:
write n in base 2 with NC digits
Compute $m \oplus n = k$, k_i ith digit
$u_i = k_i$ $i = 1,..NC$

Fig. II-5. Flow graph showing the steps to obtain the transition tableau.

CHAPTER III

DYNAMIC PROGRAMMING AND BACKWARD EQUATIONS

III.1 Preliminary Remarks

It can be easily seen that the model described in chapter II has all the properties needed for the application of dynamic programming, [13] - [16] .

In particular it is obvious that the separation property holds for the cost function, eq. (2.4). The Markovian state property is also satisfied, see section II.5. Hence the problem is:

Given the dynamic equations (2.12), find the optimal dynamic allocation strategy, using dynamic programming, to minimize the cost (2.4).

We will separate the total expected cost (2.4) in two parts

$$C = E\,\{H\,[Y(T)]\} + E \sum_{\tau=0}^{T-1} L[Y(\tau),\tau] \tag{3.1}$$

where

$$L[Y(\tau),\tau] = \sum_{i=1}^{NC} C_i y_i(\tau) + \sum_{i=1}^{NC} (1-y_l(\tau))n_i(\tau) \min_{k\epsilon I(\tau)} C_{ki} \tag{3.2}$$

is the per unit time, or immediate, cost, and

$$H\,[Y(T)] = L\,[Y(T),T] \tag{3.3}$$

is the terminal cost.

The cost to go at time t given that the system is in state i will be defined as

$$V_i(t) = E\left\{ \sum_{\tau=t}^{T} L[Y(\tau),\tau] \;\middle|\; Y(t) = i \right\} \tag{3.4}$$

and the optimal cost-to-go

$$V_i^*(t) = \min_{u(\tau)} V_i(t) \qquad i = 1,2,..M \tag{3.5}$$

From the Markovian property, the following equalities can be easily proved, see ref [11].

$$E\{L\,[Y(\tau),\tau]\;|Y(t),\; t = 0,1,..\tau\}=$$

$$E\{L\,[Y(\tau),\tau]\;|Y(\tau)\} = \tag{3.6}$$

$$\sum_{i=1}^{NC} C_i y_i(\tau) + E\left\{ \sum_{i=1}^{NC} (1-y_i(\tau))\;\theta_i(\tau) \min_{k\epsilon I(\tau)} C_{ki} \;\middle|\; Y(\tau) \right\}$$

III.2 Backwards Recursive Equations

The backwards equations for this probabilistic system can be written (see [17] pag 955) as

$$V_i^*(t) = \min_{u(t)}\{E\{L[Y(t),t]\;|Y(t)=i\} + \sum_{j=1}^{NC} P_{ij}(t,u)V_j^*(t+1) \tag{3.7}$$

$$i = 1,2,..M$$

where $P_{ij}(t,u)$ is defined as the probability of being in state j at time t+1 given control u and given that the system is in state i at time t, that is,

$$P_{ij}(t,u) = \text{Prob}\ \{Y(t+1) = j\ |Y(t) = i,u(t)\} \qquad (3.8)$$

From the expression (3.2) of the per unit time cost at time t observe that the decision u(t) at time t affects only the state Y(t+1) at (t+1) but not Y(t) and n(t) and therefore the immediate L(t) cost is control independent.

If u* is the optimal control and V* the corresponding cost to go, then:

$$V_i^*(g)=E\ \{L\ [Y(t),t]\ |Y(t)\ =i\} + \sum_{j=1}^{NC} P_{ij}(t,u^*)V_j^*(t+1) \qquad (3.9)$$

$$i = 1,2,..M$$

or in vector form

$$V^*(t) = \Lambda(t) + P(t,u^*)\ V^*(t+1) \qquad (3.10)$$

With this notation it is clear that the total minimum expected cost over the period [0,T] will be the smallest component of the vector V*(0) and, the state corresponding to this component will be the optimal initial state.

To pursue further with the investigation of the actual form of the vector $\Lambda(t)$ and matrix P(t,u*) we will begin with the cases NC=2 and NC=3, the generalization to a larger number of computers will then become apparent.

III.3 Recursive Equation for NC=2

For the case of two computers the expression of the total expected cost over the period $[0,T]$ can be written as

$$C=E\sum_{t=0}^{T}\left[\sum_{i=1}^{2} C_i y_i(t) + \sum_{i=1}^{2} (1-y_i(t))n_i(t)C_{\substack{ki\\k\neq i}}\right] =$$

$$= E\sum_{t=0}^{T}\left[C_1y_1(t)+C_2y_2(t)+(1-y_1(t))\theta_1(t)C_{21}+(1-y_2(t))\theta_2(t)C_{12}\right] \tag{3.11}$$

where we have applied (3.6) and the condition that $y_1(t) + y_2(t) \geq 1$. Therefore

$$L[Y(t),t]=C_1y_1(t)+C_2y_2(t)+(1-y_1(t))\theta_1(t)C_{21}+(1-y_2(+))\theta_2(t)C_{12} \tag{3.12}$$

From this expression we obtain immediately the components of the vector $\Lambda(t)$

$$\begin{aligned}
\lambda_1(t)&=E\{L[Y(t),t]\,|Y(t)=(0\ 1)=1\} = C_2+C_{21}\theta_1(t)\\
\lambda_2(t)&=E\{L[Y(t),t]\,|Y(t)=(1\ 0)=2\} = C_1+C_{12}\theta_2(t)\\
\lambda_3(t)&=E\{L[Y(t),t]\,|Y(t)=(1\ 1)=3\} = C_1+C_2
\end{aligned} \tag{3.13}$$

To obtain the elements of the probability matrix it is very important to follow carefully all the conditions, see Fig. II-2, imposed on the decision process. Following those rules we have obtained in Appendix A the elements of the transition matrix, as

$$P(t,u^*) = \begin{bmatrix} 1-\alpha_1^*(t)\Theta_1(t) & \varepsilon_2^*(t)\,\alpha_1^*(t)\Theta_1(t) & (1-\varepsilon_1^*(t))\alpha_1^*(t)\Theta_1(t) \\ \varepsilon_1^*(t)\alpha_2^*(t)\Theta_2(t) & 1-\alpha_2^*(t)\Theta_2(t) & (1-\varepsilon_1^*(t))\alpha_2^*(t)\Theta_2(t) \\ \varepsilon_1^*(t)\cdot(1-\varepsilon_2^*(t)) & (1-\varepsilon_1^*(t))\varepsilon_2^*(t) & (1-\varepsilon_1^*(t))(1-\varepsilon_2^*(t)) \end{bmatrix} \quad (3.14$$

It can be easily seen that in fact $P(t,u^*)$ can be obtained directly from the tableau of Fig. II-6 by the following correspondence (see Fig. III.1):

a) If the value of the control variable u_i in the tableau is

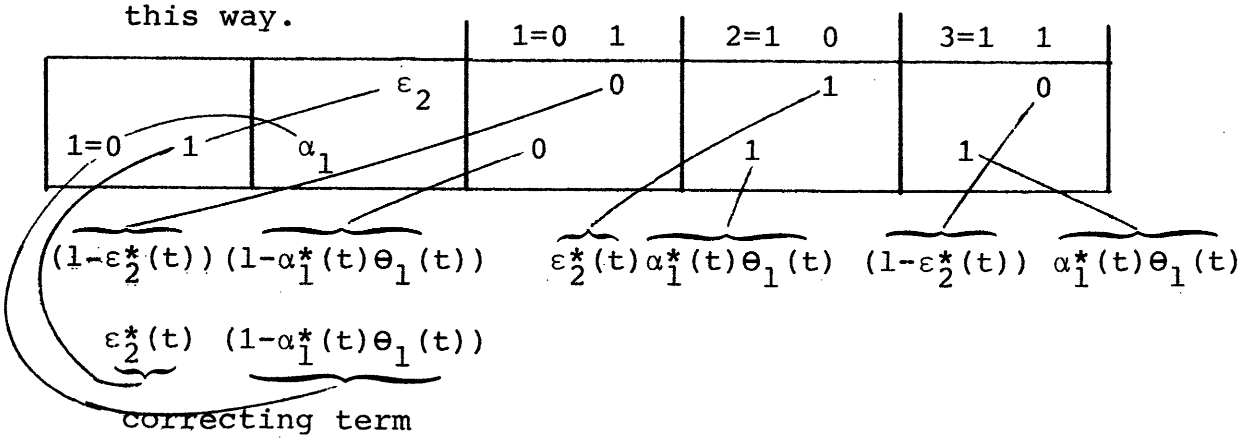
$u_i = 0$ write a term equal to $1-u_i$

$u_i = 1$ write a term equal to u_i

b) If $u_i \equiv \alpha_i$ write $\alpha_i\Theta_i$ instead of α_i

c) If the cell is on the diagonal add to the previous term a correcting term obtained considering a new cell with values given by the variables $y_1(t)$ and $y_2(t)$ and applying steps a) and b)

d) Repeat a), b) and c) for i=1 and i=2. The transition probability is the product of the two terms obtained in this way.



1=0 1
2=1 0
3=1 1
ε_2
1=0 1
α_1
0
0
1
1
0
1
$(1-\varepsilon_2^*(t))\,(1-\alpha_1^*(t)\Theta_1(t))$
$\varepsilon_2^*(t)\,\alpha_1^*(t)\Theta_1(t)$
$(1-\varepsilon_2^*(t))\;\alpha_1^*(t)\Theta_1(t)$
$\varepsilon_2^*(t)\;\;(1-\alpha_1^*(t)\Theta_1(t))$
correcting term

Fig. III-1 How to obtain the first row of the matrix from the first tableau row

This is not a surprising result and it could be easily expected from the way the tableau is constructed. Step C) is a consequence of the condition imposed that if after the arrival of the requests the optimal decision requires to erase the last copy of the system we remain in the same state. Therefore the probability of remaining in the same state (diagonal terms) has to be corrected by a term equal to the probability of requiring the erasure of the last copy. This probability is exactly the probability of going to state 0 if this state were allowed. The values of the control variables needed to go to state 0 are obtained through the "exclusive or" addition of the binary representations of present state and state 0, but this sum is always equal to the present state representation; therefore the values of the control variables are equal to the values of the allocation variables of the present state. In this way we ensure that this matrix accomplishes all the properties of a stochastic matrix, in particular the needed condition that all rows must add to one; this is so because the terms are obtained using all possible combinations of 0's and 1's with two elements (NC elements in general) and hence we always add terms like

$$A\ B+(1-A)B+A(1-B)+(1-A)(1-B) = 1 \qquad (3.15)$$

Another simplification can be obtained by observing that in every row the combination of control values that will take the system into the state $Y(t+1)=0$ is not allowed. For example in the first row of Fig. II-3, $\alpha_1=0$ $\varepsilon_2=1$ is forbidden and therefore in the first row of P(t,u) we have

$$(1-\alpha_1)\varepsilon_2 = 0 \tag{3.16a}$$

Similarly in the second row of P(t,u)

$$(1-\alpha_2)\varepsilon_1 = 0 \tag{3.16b}$$

and in the third row

$$\varepsilon_1 \; \varepsilon_2 = 0 \tag{3.16c}$$

This property will be useful sometime to simplify the expression of the transition probabilities. We have made use of this property in Appendix A.

Grouping all results together we obtain the following backward matrix equation (NC = 2)

$$\begin{bmatrix} V_1^*(t) \\ V_2^*(t) \\ V_3^*(t) \end{bmatrix} = \begin{bmatrix} C_2 + C_{21}\theta_1(t) \\ C_1 + C_{12}\theta_2(t) \\ C_1 + C_2 \end{bmatrix} +$$

$$\begin{bmatrix} 1-\alpha_1^*(t)\theta_1(t) & \varepsilon_2^*(t)\alpha_1^*(t)\theta_1(t) & (1-\varepsilon_2^*(t)\alpha_1^*(t)\theta_1(t) \\ \varepsilon_1^*(t)\alpha_2^*(t)\theta_2(t) & 1-\alpha_2^*(t)\theta_2(t) & (1-\varepsilon_1^*(t)\alpha_2^*(t)\theta_2(t) \\ \varepsilon_1^*(t) & \varepsilon_2^*(t) & 1-\varepsilon_1^*(t)-\varepsilon_2^*(t) \end{bmatrix} \begin{bmatrix} V_1^*(t+1) \\ V_2^*(t+1) \\ V_3^*(t+1) \end{bmatrix} \tag{3.17}$$

where the optimal decisions for each row of the matrix P(t,u*) are the values of the corresponding row in the tableau that give minimum scalar product with the vector V*(t + 1)

In particular, if we define

$$A=V_1^*(t+1)$$

$$B=(1-\theta_1(t))V_1^*(t+1)+\theta_1(t)V_2^*(t+1) \tag{3.18}$$

$$C=(1-\theta_1(t))V_1^*(t+1)+\theta_1(t)V_3^*(t+1)$$

then if the system is in state 1 at time t

$$\begin{cases}\alpha_1^*(t) = 0\\ \varepsilon_2^*(t) = 0\end{cases} \quad \text{if } A < B \text{ and } A < C \tag{3.19a}$$

$$\begin{cases}\alpha_1^*(t) = 1\\ \varepsilon_2^*(t) = 1\end{cases} \quad \text{if } B < A \text{ and } B < C \tag{3.19b}$$

$$\begin{cases}\alpha_1^*(t) = 1\\ \varepsilon_2^*(t) = 0\end{cases} \quad \text{if } C < A \text{ and } C < B \tag{3.19c}$$

In the same way the optimal decisions being in state 2 and 3 can be obtained.

We will see some numerical applications of these equations in chapter V.

III-4. Recursive Equations for NC=3 and Generalization to any NC

For NC=3 the total expected cost over the period [0,T] can be expressed as (remember $C_{ii} = 0, \forall i$)

$$C = E \sum_{t=1}^{T} \left[\sum_{i=1}^{3} c_i y_i(t) + \sum_{i=1}^{3} \sum_{j\geq i}^{3} \sum_{\substack{k\neq i\\ k\neq j}} y_i(t) y_j(t) n_k(t) \min_{l\in\{i,j\}} C_{lk} \right]$$

$$= E \sum_{t=1}^{T} L[Y(t),t] \tag{3.20}$$

The components of the vector $\Lambda(t)$ are obtained as

$$\lambda_1(t)=E\ \{L[Y(t),t]\,|Y(t)=(0\ 0\ 1)\ =\ 1\}\ =\ C_3+C_{31}\theta_1(t)+C_{32}\theta_2(t)$$

$$\lambda_2(t)=E\ \{L[Y(t),t]\,|Y(t)=(0\ 1\ 0)\ =\ 2\}\ =\ C_2+C_{21}\theta_1(t)+C_{23}\theta_3(t)$$

$$\lambda_3(t)=E\ \{L[Y(t),t]\,|Y(t)=(0\ 1\ 1)\ =\ 3\}\ =\ C_2+C_3+\theta_1(t)\min\{C_{21},C_{31}\}$$

$$\text{" " " " "}\ \ ---\ \ --\ \ \text{etc.} \tag{3.21}$$

The whole vector is

$$\Lambda(t) = \begin{bmatrix} C_3 + C_{31}\ \theta_1(t) + C_{32}\ \theta_2(t) \\ C_2 + C_{21}\ \theta_1(t) + C_{23}\ \theta_3(t) \\ C_2 + C_3 + \theta_1(t)\ \min\ (C_{21},\ C_{31}) \\ C_1 + C_{12}\ \theta_2(t) + C_{13}\ \theta_3(t) \\ C_1 + C_3 + \theta_2(t)\ \min\ (C_{12},\ C_{32}) \\ C_1 + C_2 + \theta_3(t)\ \min\ (C_{13},\ C_{23}) \\ C_1 + C_2 + C_3 \end{bmatrix} \tag{3.22}$$

The way to construct these components from the state vector is simple.

The easy rules are sketched in the flow chart of Fig. III-2.

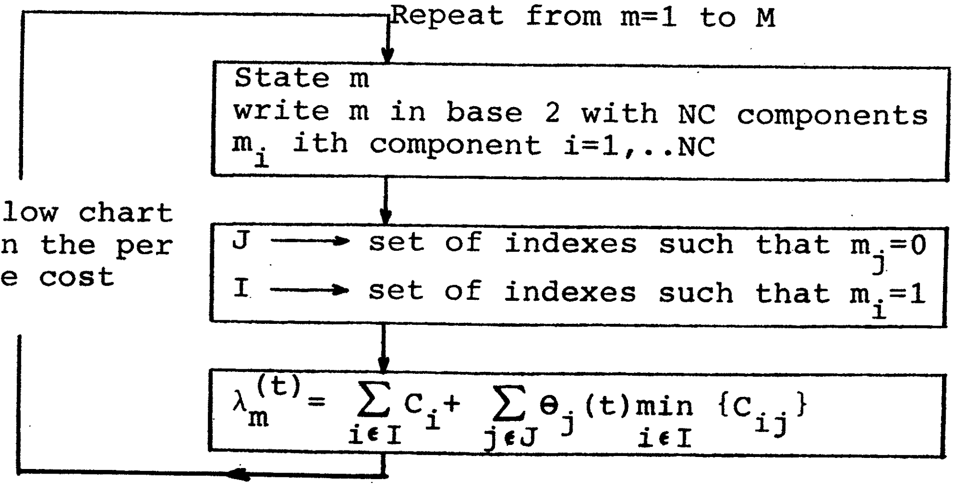

Repeat from m=1 to M
State m
write m in base 2 with NC components
m_i ith component i=1,..NC
J ⟶ set of indexes such that $m_j=0$
I ⟶ set of indexes such that $m_i=1$
$\lambda_m^{(t)} = \sum_{i\in I} C_i + \sum_{j\in J} \theta_j(t) \min_{i\in I} \{C_{ij}\}$

Fig. III.2 Flow chart to obtain the per unit time cost vector

As far as the probability transition matrix is concerned, we can calculate easily its components by making use of the rules stated for the case NC=2 and the tableau of Fig. II-7. Some of the components are shown below (for brevity we delete the variable t).

$$P(t,u) = \begin{bmatrix} (1-\alpha_1\theta_1)(1-\alpha_2\theta_2) & (1-\alpha_1\theta_1)\alpha_2\theta_2\varepsilon_3 & \cdots \\ (1-\alpha_1\theta_1)\varepsilon_2\alpha_3\theta_3 & (1-\alpha_1\theta_1)(1-\alpha_3\theta_3) & \cdots \\ (1-\alpha_1\theta_1)\varepsilon_2(1-\varepsilon_3) & (1-\alpha_1\theta_1)(1-\varepsilon_2)\varepsilon_3 & \cdots \\ \vdots & \vdots & \\ \varepsilon_1\varepsilon_2 & \varepsilon_1\varepsilon_3 & \cdots \end{bmatrix} \qquad (3.23)$$

where we have applied a property similar to (3.16) so that for example in the 7th row of P given in (3.23) we have $\varepsilon_1\varepsilon_2\varepsilon_3 = 0$.

It can be seen now that the rules we developed in constructing the immediate cost vector and the transition probability matrix for NC=2 and NC=3, generalize easily for a network of arbitrary size. These rules will allow for an easy algorithm to be implemented on a computer. To make things concrete, we illustrate this in the following example:

Example:

Suppose we have a network with five computers NC=5, and being in state 3 at time t, we want to know the immediate cost and the probability of being in state 17 at time t+1.

First of all we write the vector representation of state 3 and its control variables:

$$Y(t) = 3 = (0\ 0\ 0\ 1\ 1) \longrightarrow (\alpha_1,\alpha_2,\alpha_3,\varepsilon_4,\varepsilon_5) \qquad (3.24)$$

From this representation we can immediately write the per unit time cost.

$$\lambda_3(t) = C_4+C_5+\Theta_1(t) \min (C_{41}, C_{51}) + \Theta_2(t) \min (C_{42}, C_{52}) +\Theta_3(t) \min (C_{43}, C_{53}) \qquad (3.25)$$

To obtain $P_{3,17}(t)$ we also need the vector representation of state 17

$$Y(t+1) = 17 = (1\ 0\ 0\ 0\ 1) \qquad (3.26)$$

the value of the control variables we need for this transition are:

$$(0\ 0\ 0\ 1\ 1) \oplus (1\ 0\ 0\ 0\ 1) = (1\ 0\ 0\ 1\ 0) \equiv (\alpha_1\alpha_2\alpha_3\varepsilon_4\varepsilon_5) \qquad (3.27)$$

therefore

$$\overset{3}{(0\ 0\ 0\ 1\ 1)} \longrightarrow \begin{array}{|c|c|c|c|c|}\hline \alpha_1 & \alpha_2 & \alpha_3 & \varepsilon_4 & \varepsilon_5 \\ \hline 1 & 0 & 0 & 1 & 0 \\ \hline \end{array} \longrightarrow \overset{17}{(1\ 0\ 0\ 0\ 1)} \qquad (3.28)$$

now we can write that the transition probability is

$$P_{3,17}=\alpha_1\Theta_1(1-\alpha_2\Theta_2)(1-\alpha_3\Theta_3)\varepsilon_4(1-\varepsilon_5) \qquad (3.29)$$

It can be useful to verify that in fact we will arrive to the same expression if this probability is computed by a straight forward calculation.

From the discussion in section II-6 and Fig. II-2 we see that we can begin in state 3 and finish in state 17 in four different ways:

1) We decide to go to (1 0 0 0 1) ≡ 17 and there is a request from computer 1

decision

α_1	α_2	α_3	ε_4	ε_5
1	0	0	1	0

$$\text{Prob}\{\text{ Pos. } 1\} = \alpha_1(1-\alpha_2)(1-\alpha_3)\varepsilon_4(1-\varepsilon_5) \text{ Prob }\{n_1=1\} =$$

$$=\alpha_1(1-\alpha_2)(1-\alpha_3)\varepsilon_4(1-\varepsilon_5)\theta_1 \tag{3.30}$$

2) We decide to go to (1 1 0 0 1)≡25 and there is a request from computer 1 but there is no request from computer 2

decision = (1 1 0 1 0)

$$\text{Prob}\{\text{ Pos. } 2\} = \alpha_1\alpha_2(1-\alpha_3)\varepsilon_4(1-\varepsilon_5)\theta_1(1-\theta_2) \tag{3.31}$$

3) We decide to go to (1 0 1 0 1) ≡ 21 and there is no request from computer 3 but we have a request from computer 1:

decision = (1 0 1 1 0) (3.32)

$$\text{Prob}\{\text{ Pos. } 3\} = \alpha_1(1-\alpha_2)\alpha_3\varepsilon_4(1-\varepsilon_5)\theta_1(1-\theta_3)$$

4) We decide to go to (1 1 1 0 1) ≡ 29 but there is no request from 2 and 3 and we have request from 1

decision = (1 1 1 1 0)

$$\text{Prob}\{\text{Pos.}4\} = \alpha_1\alpha_2\alpha_3\varepsilon_4(1-\varepsilon_5)\theta_1(1-\theta_2)(1-\theta_3) \tag{3.33}$$

As we can see these four possibilities are the results of the following four decisions

α_1	α_2	α_3	ε_4	ε_5	
1	0	0	1	0	with prob. θ_1
1	1	0	1	0	with prob. $\theta_1(1-\theta_2)$
1	0	1	1	0	with prob. $\theta_1(1-\theta_3)$
1	1	1	1	0	with prob. $\theta_1(1-\theta_2)(1-\theta_3)$

(3.34)

Adding up those four probabilities we have

$$\mathrm{Prob}\{\mathrm{Pos.1}\}+\mathrm{Prob}\{\mathrm{Pos.}\ 2\}+\mathrm{Prob}\{\mathrm{Pos.}\ 3\}+\mathrm{Prob}\{\mathrm{Pos.}\ 4\}$$

$$\begin{aligned}
&= \alpha_1 (1-\alpha_2)(1-\alpha_3)\ \varepsilon_4\ (1-\varepsilon_5)\ \theta_1 + \\
&+ \alpha_1\ \alpha_2\ (1-\alpha_3)\ \varepsilon_4\ (1-\varepsilon_5)\ \theta_1(1-\theta_2) + \\
&+ \alpha_1\ (1-\alpha_2)\ \alpha_3\ \varepsilon_4\ (1-\varepsilon_5)\theta_1(1-\theta_3) + \\
&+ \alpha_1\ \alpha_2\ \alpha_3\ \varepsilon_4\ (1-\varepsilon_5)\theta_1(1-\theta_2)(1-\theta_3) = \\
&=\alpha_1\theta_1\varepsilon_4(1-\varepsilon_5)\Big[(1-\alpha_2)(1-\alpha_3)+\alpha_2(1-\alpha_3)(1-\theta_2)+ \\
&\qquad +(1-\alpha_2)\ \alpha_3\ (1-\theta_3)+\alpha_2\ \alpha_3\ (1-\theta_2)(1-\theta_3)\Big] = \\
&=\alpha_1\theta_1\varepsilon_4\ (1-\varepsilon_5)(1-\alpha_2\theta_2)(1-\alpha_3\theta_3) = P_{3,17} \qquad (3.35)
\end{aligned}$$

as was obtained in (3.29)

We could have written the remaining probabilities in the transition matrix in the same way as we did for $P_{3,17}$. Therefore in order to analyze any network under the conditions stated in chapters I and II we only have to build up the recursive equations, using the rules described before and move backward in time until we reach the steady state, or arrive at t=0.

Nevertheless while implementing the dynamic programming procedure we do not need to calculate all the probabilities of the transition matrix. As it will be seen below the reason is that after the control values are decided upon many of the terms will be known to be zero. For instance, consider the case of the above example, in which we were in state 3 and the decision was "go to state 17". The only probabilities that will be different from 0 in the 3rd row of the transition matrix are $P_{3,17}$ and $P_{3,1}$ where

$$\begin{aligned} 3 &= (0\ 0\ 0\ 1\ 1) \\ u &= (1\ 0\ 0\ 1\ 0) = (\alpha_1\alpha_2\alpha_3\varepsilon_4\varepsilon_5) \qquad (3.36) \\ 17 &= (1\ 0\ 0\ 0\ 1) \\ 1 &= (0\ 0\ 0\ 0\ 1) \end{aligned}$$

The reason is that the only condition needed to accomplish the decision is having a request from computer 1, the only computer in the decision vector with a control variable α equal to 1. If this request does not come the system will move to state 1 (we only excute $\varepsilon_4 = 1$) and there is no possibility to go to any other state with that decision vector.

This rule can be easily generalized. Being in state n and having made decision u(t), the only probabilities different from zero in row n of P(t,u) are the probabilities corresponding to destination states resulting from applying to state vector n the decision vectors obtained from vector u(t) making all possible substitutions of 0's and 1's in places where there are copying variables (α's) equal to 1 in u(t).

For instance, if n=3 as before, but now u(t) = (1 1 0 1 0) we will have

$$\overset{3}{(0\ 0\ 0\ 1\ 1)} \longrightarrow \boxed{\begin{array}{c} \alpha_1\alpha_2\alpha_3\varepsilon_4\varepsilon_5 \\ \hline 1\ 1\ 0\ 1\ 0 \end{array}} \longrightarrow \overset{25}{(1\ 1\ 0\ 0\ 1)} \quad (3.37)$$

Then, in order to implement this decision,requests from computers 1 and 2 are needed, and therefore we have the following possibilities

	α_1	α_2	α_3	ε_4	ε_5	
$\overset{3}{(0\ 0\ 0\ 1\ 1)} \rightarrow$	1	1	0	1	0	$\rightarrow$ (1 1 0 0 1) = 25 with Prob $\theta_1\theta_2$
	1	0	0	1	0	$\rightarrow$ (1 0 0 0 1) = 17 with Prob $\theta_1(1-\theta_2)$
	0	1	0	1	0	$\rightarrow$ (0 1 0 0 1) = 9 with Prob $(1-\theta_1)\theta_2$
	0	0	0	1	0	$\rightarrow$ (0 0 0 0 1) = 1 with Prob $(1-\theta_1)(1-\theta_2)$

If ε_5 would have had the value 1 instead of 0 the last transition will go to state 3, the starting state, in order to avoid the erasure of the last copy.

A schematic way showing how to compute the transition probabilities using these rules is shown in Fig. II-3. A flow-chart showing how to compute row n, of the probability transition matrix, when the decision is "go to state m", is shown in Fig. III.3 b). In the flow-chart we assume that we have available a subroutine called BITS such that given n, a number, and NC number of components it returns the base 2 representation of n with NC components. The calling sequence will be

CALL BITS (n, NC, NB2)

Furthermore, we assume we also have available the function

L = DECI(k, LBk, NC)

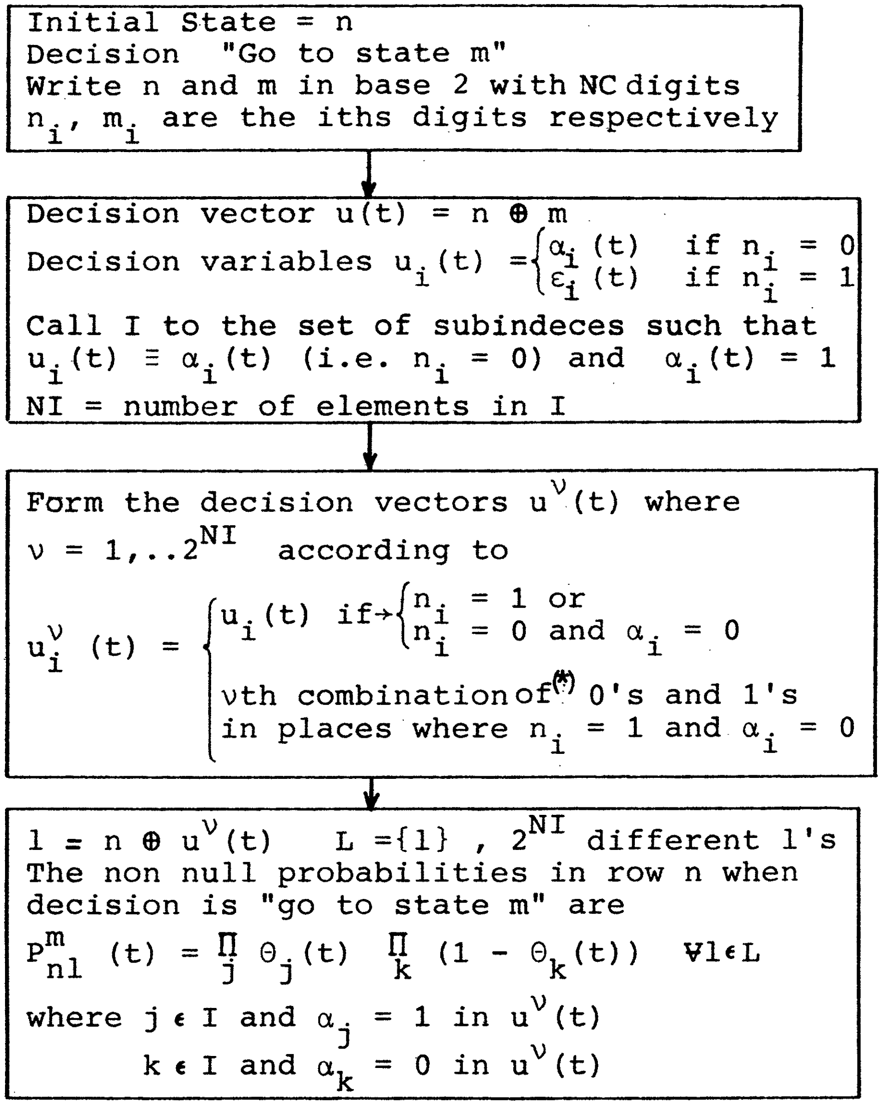
such that given a vector LBk with NC components it returns the number L whose representation in base k is LBk.

These subroutines are given in Appendix C.

The simplifications explained so far can produce a great saving in computation because, for instance, in the first case presented, only 2 of the $2^5-1 = 31$ components are different from zero.

The optimization procedure for each starting state will consist then in the computation of the non null probabilities for the initial state row for every possible transition; taking scalar product of these non null probabilities by the corresponding costs - to - go and choosing the smallest result. The decision giving place to the smallest product is the optimal decision for that initial state and the product added to the per unit time cost for that state will produce the next (backward) cost - to - go.

The flow chart of fig. III-4 shows the set of operations involved in the optimization process. In the next section we will show how this model can be easily extended to problems with constraints in the state space.



Initial State = n
Decision "Go to state m"
Write n and m in base 2 with NC digits
n_i, m_i are the iths digits respectively

Decision vector $u(t) = n \oplus m$

Decision variables $u_i(t) = \begin{cases} \alpha_i(t) & \text{if } n_i = 0 \\ \varepsilon_i(t) & \text{if } n_i = 1 \end{cases}$

Call I to the set of subindeces such that $u_i(t) \equiv \alpha_i(t)$ (i.e. $n_i = 0$) and $\alpha_i(t) = 1$
NI = number of elements in I

Form the decision vectors $u^\nu(t)$ where $\nu = 1,..2^{NI}$ according to

$$u_i^\nu(t) = \begin{cases} u_i(t) \text{ if} \rightarrow \begin{cases} n_i = 1 \text{ or} \\ n_i = 0 \text{ and } \alpha_i = 0 \end{cases} \\ \nu\text{th combination of}^{(*)} \text{ 0's and 1's} \\ \text{in places where } n_i = 1 \text{ and } \alpha_i = 0 \end{cases}$$

$l = n \oplus u^\nu(t)$ $L = \{l\}$, 2^{NI} different l's
The non null probabilities in row n when decision is "go to state m" are

$$P_{nl}^m(t) = \prod_j \theta_j(t) \prod_k (1 - \theta_k(t)) \quad \forall l \epsilon L$$

where $j \epsilon I$ and $\alpha_j = 1$ in $u^\nu(t)$
$k \epsilon I$ and $\alpha_k = 0$ in $u^\nu(t)$

Fig. III-3 a) Flow-chart showing how to construct the non-null probabilities of row n when decision is "go to state m"

....

*The first combination is 00---0 and the last one is 111--1.

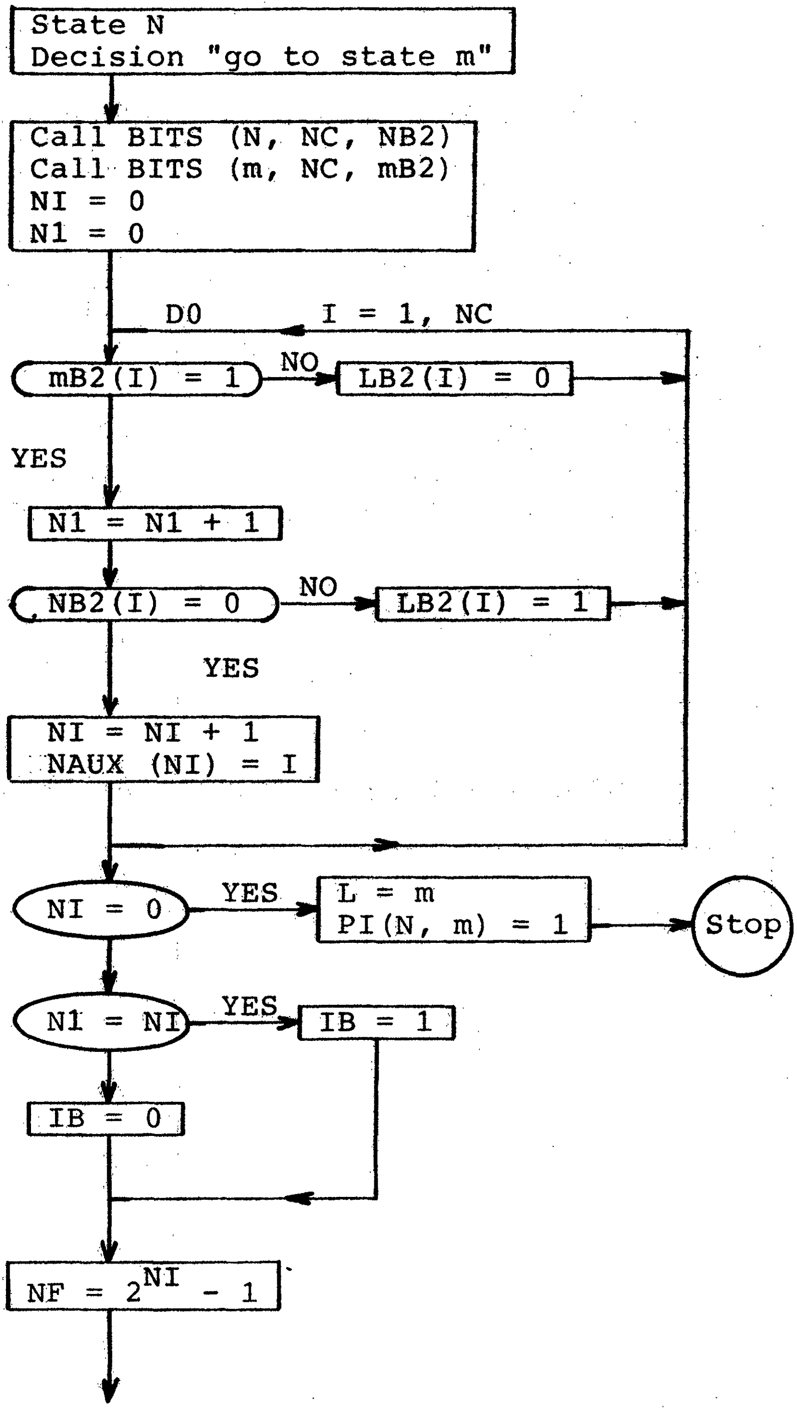


Fig. III.3 b.

(continued on next page)

(cont'd)

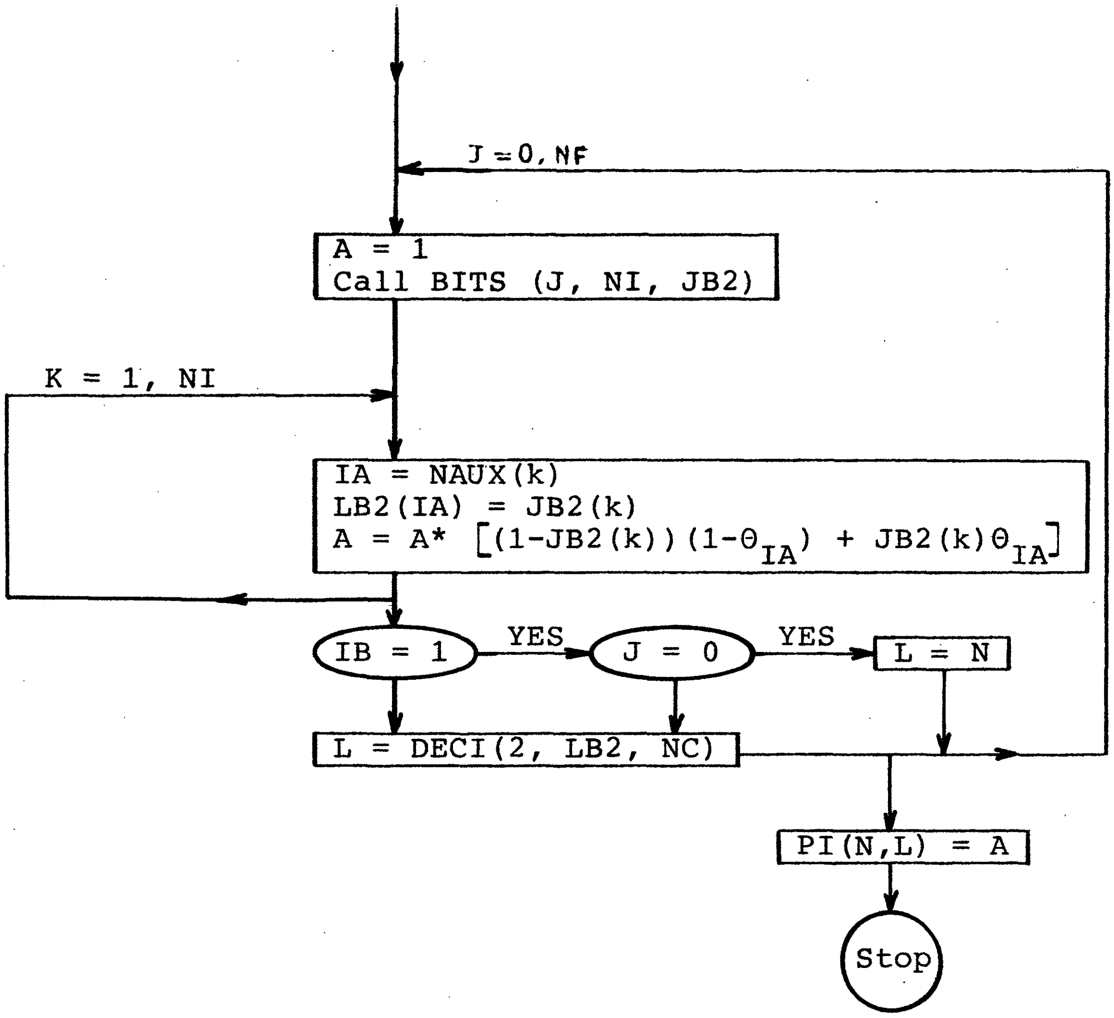


Fig. III.3 b) Fortran Flow Chart of Fig. III.3 a.

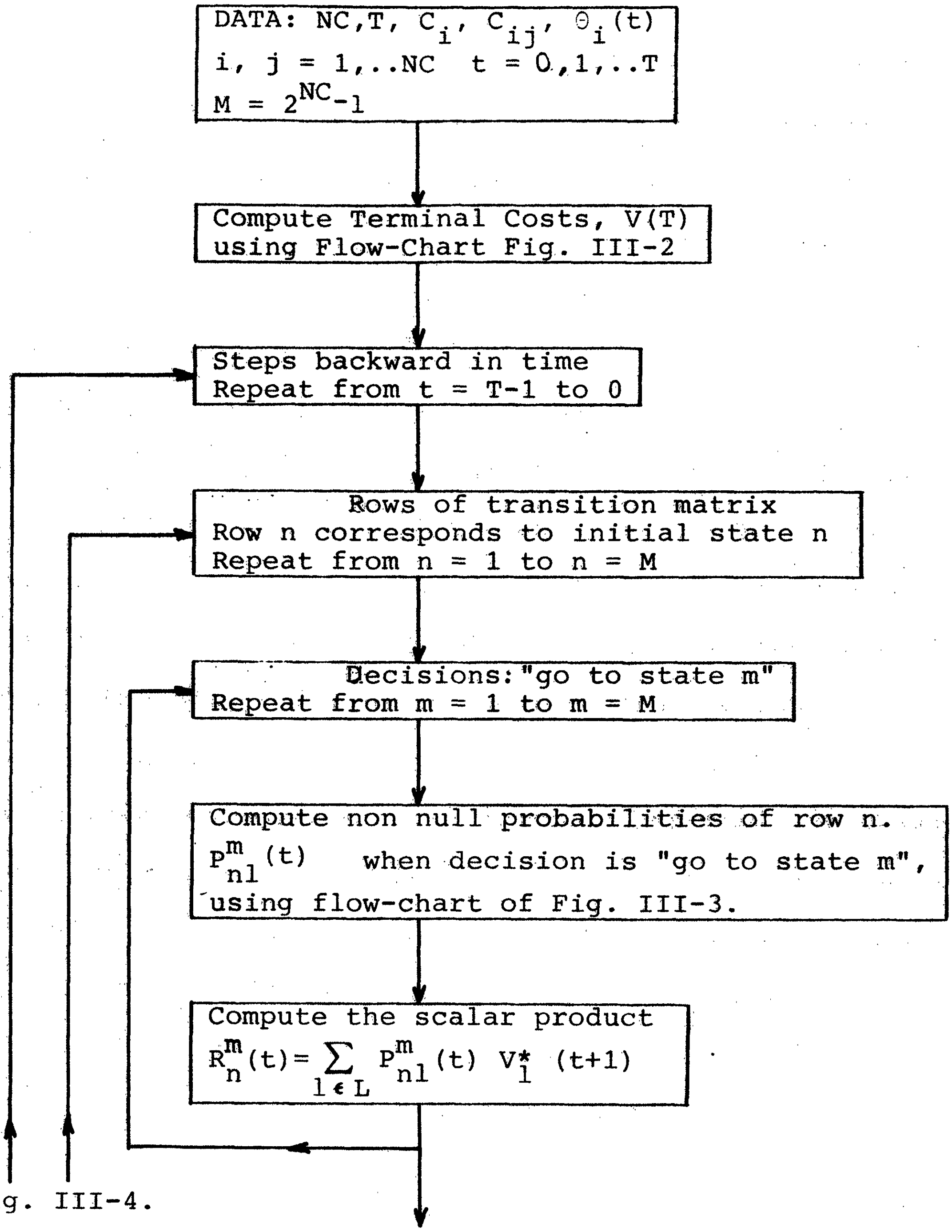

DATA: NC, T, C_i, C_{ij}, $\Theta_i(t)$
$i, j = 1, ..NC$ $t = 0, 1, ..T$
$M = 2^{NC}-1$
Compute Terminal Costs, V(T)
using Flow-Chart Fig. III-2
Steps backward in time
Repeat from t = T-1 to 0
Rows of transition matrix
Row n corresponds to initial state n
Repeat from n = 1 to n = M
Decisions: "go to state m"
Repeat from m = 1 to m = M
Compute non null probabilities of row n.
$P^m_{nl}(t)$ when decision is "go to state m",
using flow-chart of Fig. III-3.
Compute the scalar product
$R^m_n(t) = \sum_{l \in L} P^m_{nl}(t)\ V^*_l\ (t+1)$

Fig. III-4.

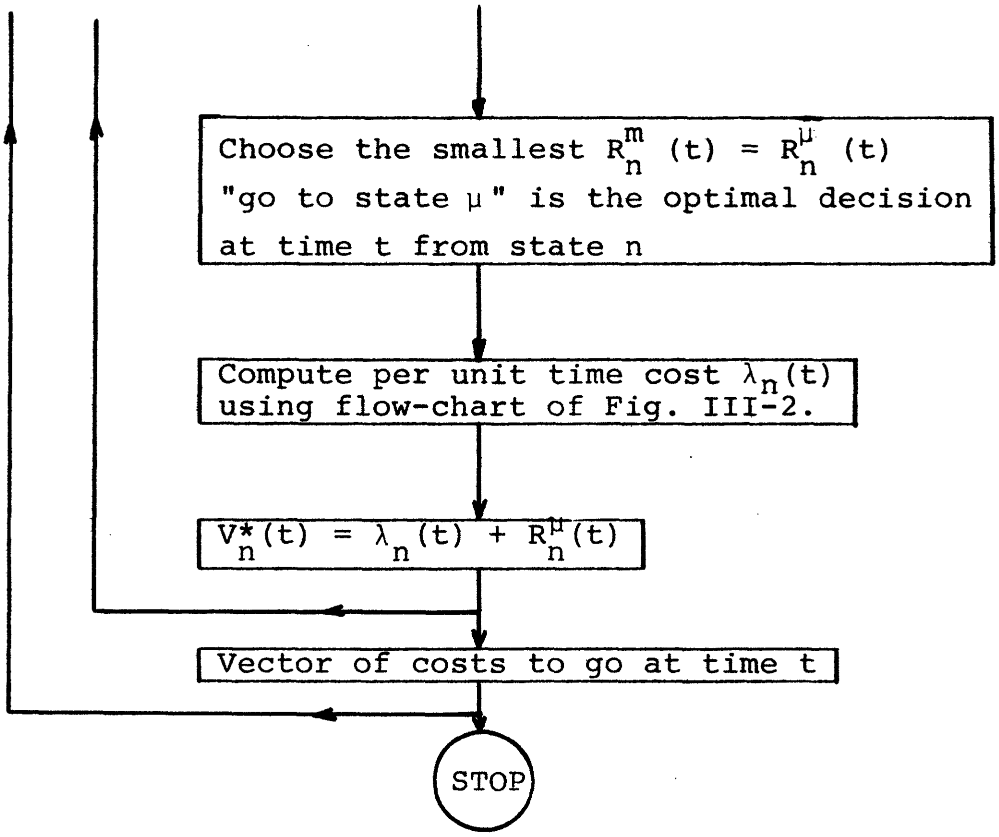

Choose the smallest $R_n^m(t) = R_n^{\mu}(t)$
"go to state μ" is the optimal decision
at time t from state n
Compute per unit time cost $\lambda_n(t)$
using flow-chart of Fig. III-2.
$V_n^*(t) = \lambda_n(t) + R_n^{\mu}(t)$
Vector of costs to go at time t
STOP

Fig. III.4. Flow chart of the Optimization Process.

III-5. Constraints in the State Space

The problem formulation and the model described will allow us to handle easily constraints in the state space. These constraints may take the form of a maximum number of copies allowed in the system at any instant of time or, not allowing copies of the file simultaneously in two or more given computers.

For instance, if given a network with three computers (NC=3), three copies are not allowed in the system simultaneously then state 7 will be taken out of the set of admissible states; if on the other hand, the restriction is that there cannot be copies simultaneously in computers 1 and 2, then state 6 is taken out of the state space.

One example of these types of constraints was presented before when state 0 was not allowed. Therefore, unallowed states will be treated here in the same way state 0 was treated before. To gain some insight we present an example:

Consider a network with four computers (NC=4). If the present state is 1 = (0 0 0 1) and the decision is u(t) = (1 1 1 0), then the intended state is 15 = (1 1 1 1).

$$(0\ \ \overset{1}{0}\ \ 0\ \ 1) \longrightarrow \begin{array}{|cccc|}\hline \alpha_1 & \alpha_2 & \alpha_3 & \varepsilon_4 \\ \hline 1 & 1 & 1 & 0 \\ \hline \end{array} \longrightarrow (1\ \ \overset{15}{1}\ \ 1\ \ 1)$$

If there is a request from computers 1 and 3 but not from computer 2, the system will go to state (1 0 1 1) = 11 and this event will occur with probability $\theta_1(1-\theta_2)\theta_3$.

Suppose now that state 11 is not allowed, then another decision has to be made. The situation is such that any state of the form (a o b c) can be reached where a, b and c can be 0 or 1 (but not all three equal to 0). Nevertheless, considering so general a decision at this point will make the problem very complicated, therefore a decision to remain in the same state will be considered. This is a particular case of the whole set of possible decisions. This kind of decision will give to state 11 the same treatment as to state 0, as suggested, before. The probability of remaining in one state will now be composed of the following terms:

$$\text{Prob}\{\text{remain in same state}\} = \text{Prob}\{\text{going to this state}\} + \text{Prob}\{\text{going to state } 0\} + \text{Prob}\{\text{going to not allowed states}\} \tag{3.39}$$

In the general algorithm we will eliminate the rows and columns corresponding to unallowed states and add their probabilities to the diagonal terms. It should be noticed that some extended simplification properties, similar to the one obtained in (3.16), could be obtained from the new unallowed states.

For instance, for the example above where state 1 was the current state and state 11 was not allowed, we have that

$$\alpha_1(1-\alpha_2)\alpha_3(1-\varepsilon_4) = 0 \tag{3.40}$$

if, moreover, state (1 0 0 1) = 9 were not allowed as well, then

$$\alpha_1(1-\alpha_2)(1-\alpha_3)(1-\varepsilon_4) = 0 \tag{3.41}$$

and from these two equalities

$$\alpha_1(1-\alpha_2)(1-\varepsilon_4) = 0 \qquad (3.42)$$

An illustrative example where these facts are applied is studied in Appendix B for the case of a network with two computers (NC=2). It is shown there that if we restrict the system to have only one copy at any instant of time the backward equations simplify to the equations given by A. Segall in [11], where the restriction was to operate with only one copy.

CHAPTER IV

UPDATING TRAFFIC AND NODE FAILURES

IV-1. Updating Traffic

The updating traffic consists of requests generated at some nodes after a request of the file, with the only purpose of modifying, partially or completely, the content of the file. With this definition it is seen that the updating information generated at any node, should be sent to all other nodes that possess a copy of the file.

It will be assumed, for the present study, that,

1) This kind of traffic is generated at any node as a fraction, of the query traffic of this particular node. In general these fractions can be time dependent variables. If we denote them by

$$\rho_1(t),\ \rho_2(t),\ --\ \rho_{NC}(t)$$

the rate of updating traffic generated from node i at time t will be then

$$\rho_i(t)\theta_i(t)$$

2) The updating traffic is implemented before the decision has been activated but after the request has taken place. The sequence of events is represented in fig. IV-1 as a generalization of fig. II-2.

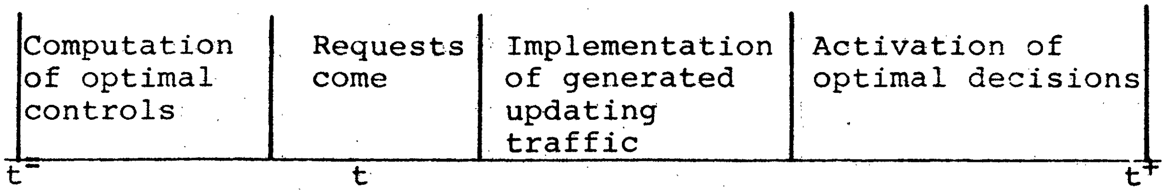

Computation of optimal controls
Requests come
Implementation of generated updating traffic
Activation of optimal decisions
t^-
t
t^+

Fig. IV-1. Sequence of events including updating traffic

With probability $\rho_i(t)$ an updating of the file is generated at any computer i that requested the file and is sent to all computers that will keep a copy at time t+1.

3) We will assume that there is no conflict between the updating commands coming from different computers. This is sometimes a serious problem in a practical case because it can force us to block requests, while some updating is being, done in order to avoid the processing of some old, and then useless or even conflictive, information.

Under assumptions 1) and 2) we can say that updating traffic is not a function of present state, but only of present rates and subsequent states (as we will see in sections IV-2 and IV-3, this property is not true if we include the possibility of node failures). The only change to take place in the recursive equations will be in vector V*(t+1) that will have some extra terms added to its components. The new costs-to-go vector at time t+1 will be now

$$\begin{bmatrix} V_1^*(t+1) + \sum\limits_{k=1}^{N-1} \rho_k(t)\ \theta_m(t)\ C_{kN} \\ \\ V_2^*(t+1) + \sum\limits_{\substack{k=1 \\ k\neq N-1}}^{N} \rho_k(t)\theta_k(t)\ C_{k,N-1} \\ \vdots \\ V_j^*(t+1) + \sum\limits_{k=1}^{N} \rho_k(t)\theta_k(t) \sum\limits_{\substack{l\in I(j) \\ l\neq k}} C_{kl} \\ \vdots \\ V_M^*(t+1) + \sum\limits_{k=1}^{N} \rho_k(t)\ \theta_k(t) \sum\limits_{\substack{l=1 \\ l\neq k}}^{N} C_{kl} \end{bmatrix} = V^*(t+1)+R(t) \tag{4.1}$$

Here I(j) is the computers containing a copy at state j or in other words the set of subscripts corresponding to 1's in vector state j. We are assuming here that the only charge involved in updating a copy at computer i by computer j is the transmission cost C_{ij}.

The recursive equation will now be

$$V^*(t) = \Lambda(t) + P(t,u^*)\left[V^*(t+1) + R(t)\right] \tag{4.2}$$

where R(t) is an M-components column vector. Several examples investigating how the updating traffic affects the dynamic control of the system will be studied in Chapter V.

IV-2. Network with Node Failures

In this section the necessary changes in the model to include the problem of node failures will be considered.

We shall assign to every computer a probability, P_f, of failure and a probability, P_r, of recovery according to the following definitions

$$P_f \triangleq \text{Prob. of failure per unit of time} \tag{4.3}$$

$$P_r = \text{Prob. of recovery per unit of time given that the computer is out of order}$$

It is assumed that P_f and P_r are independent and the same for all computers and instant of time, or in other words that the failure and recovery processes are modeled as two independent Bernoulli processes with rates P_f and P_r respectively.

Under these circumstances, the new state has to carry along information about two facts

1) the computer condition (working or not) and,
2) if the computer is on, whether or not the computer has a copy of the file

Therefore it can be concluded that each component of the vector state has to bring information of one of three possibilities

a) computer out of order

b) computer working and without a copy of the file

c) computer working and with a copy of the file

If we represent possibilities b) and c) with a 0 and 1 respectively, as before, and possibility a) with the digit 2, the new state vector will be equivalent to a base 3 representation of some decimal number. Since this representation is unique we see that we can name the states by the decimal number whose base 3 representation is the NC-dimensional vector. For example

$Y(t) = (0\ 2\ 1\ 1)$

will correspond to the state $1 + 3 + 2 \times 3^2 = 22$ of a network with 4 computers, where computer 2 is out of work, computers 1, 3 and 4 are working and the last two have a copy of the file in their memories.

Our model will now contain the following further assumptions:

a) When a computer is restored, it comes up with no copy in its memory. This says that no computer can make a transition from state 2 to state 1.

b) If there is at least one computer, say i, in working condition but there are no copies in the system then one copy is brought from outside (special memory) at a price C_{oi}. If there are several computers in working condition with no copy in their memories, the system will bring a copy from outside to the computer with the smallest C_{oi}. Obviously

$$C_{oi} >> C_{ij} \qquad i,j = 1,2,..NC$$

and the quantities C_{oi} will carry a measure of the risk of losing all copies that we are willing to take.

c) The time between the points t^- and t^+ of Fig. II-2 is very small compared to the unit interval (t, t+1). For our purpose that means that the probability of a failure in the interval (t^-, t^+) is negligible.

With these assumptions the number of states in the state space will be the number of different NC-dimensional vectors that can be formed with 3 digits, that is, $M=3^{NC}$.

In the present case state 0 is in the state space, because the system can go to this state after being at state M-1, when all the computers are inoperative, provided that all computers become operative in only one interval of time. The decision variables will remain the same as before except that there are no decision variables for inoperative computers. That is, there are not decision variables for the components of the state vector with value equal to 2. In particular, when all computers are not operative, there is no decision to be made. The only thing to do is to wait until one or more computers recover and then bring a copy into the system from outside.

Given the state at time t, the transition to time (t+1) will be obtained as follows

a) Decide upon the value of the control variables (if any)

b) Perform "exclusive or" of the control variables with the actual requests and modify accordingly the state variables (as in section II-6)

c) The failure or recovery of computers (if any) will modify in turn the former transition.

The state at time (t+1) will then be the result of the above three operations.

In following sections we apply these concepts to the case NC=2, NC=3 and show how they generalize to any NC.

IV.3. Recursive Equations for NC=2 considering Node Failures

The states per NC=2 are

$$\begin{array}{llll} Y(t) & 0=(0\ 0) & Y(t)= & 5=(1\ 2) \\ & 1=(0\ 1) & & 6=(2\ 0) \\ & 2=(0\ 2) & & 7=(2\ 1) \\ & 3=(1\ 0) & & 8=(2\ 2) \\ & 4=(1\ 1) & & \end{array} \qquad (4.4)$$

Let C_{01} be the costs of bringing copies from outside to computers 1 and 2 respectively. Assuming $C_{01} < C_{02}$ the per-unit-time costs are

$$\begin{array}{ll} \lambda_0(t) = C_{01} & \lambda_5(t) = C_1 \\ \lambda_1(t) = C_2 + C_{21}\theta_1(t) & \lambda_6(t) = C_{02} \\ \lambda_2(t) = C_{01} & \lambda_7(t) = C_2 \\ \lambda_3(t) = C_1 + C_{12}\theta_2(t) & \lambda_8(t) = 0 \\ \lambda_4(t) = C_1 + C_2 & \end{array} \qquad (4.5)$$

It seems, from these values of the costs, that the optimization process will try to keep the system always at state 8, because this state has the smallest cost, but the decision "go to state 8" is not among the set of admissible decisions, since this will erase all copies from the system.
For the case NC=2 the only states that will generate control variables are states 1, 3 and 4 and clearly these control variables will give rise to transitions among these states only

If we represent by:

⊠ —— the transitions ruled by control variables (when no failures or recoveries are involved)

X —— the transitions due to some failure or recovery of some computer

* —— the transitions due to a forced decision (namely a copy has to be brought from outside)

The following tableau of possible transitions can be sketched (remember $C_{01} < C_{02}$)

	states								
States	0	1	2	3	4	5	6	7	8
0				*		X	X		X
1		⊠	X	⊠	⊠	X	X	X	X
2				X		*	X		X
3		⊠	X	⊠	⊠	X	X	X	X
4		⊠	X	⊠	⊠	X	X	X	X
5				X		*	X		X
6		X	X					*	X
7		X	X					*	X
8	X		X				X		X

State	Comps	
	1	2
0	0	0
1	0	1
2	0	2
3	1	0
4	1	1
5	1	2
6	2	0
7	2	1
8	2	2

(4.6)

The empty entries in the tableau correspond to impossible transitions; this is an important difference with respect to the tableau with no failures where all transitions are possible. These empty places will generate zeroes in the transition matrix and this property will be useful later to reduce the amount of computation in the optimization process.

Let us calculate the probability transition for the entries of the tableau:

<u>row 0</u>: initial state 0 ≡ (0 0)

With the condition $C_{01} < C_{02}$, there will be a transition from state 0 to state 3 if there are no failures. That is,

$$P_{0\ 0}(t) = (1-P_f)^2 \tag{4.7}$$

With only one failure the transition will be to states 5 or 6 depending on the failed computer and the probabilities will be

$$P_{0\ 5}(t) = P_{0\ 6}(t) = P_f(1-P_f) \tag{4.8}$$

If there are two failures

$$P_{0\ 8}(t) = P_f^2 \tag{4.9}$$

Except for the above, no other transitions are possible.

<u>row 1</u>: initial state 1 ≡ (0 1)

The transitions to 1,3 and 4 are controled by the decision variables defined in chapters II and III and the condition that no failures occur. Denoting by $P^{II}_{ij}(t)$ the probability of going from state i to state j for the case of two computers and no failures, the following transition probabilities can be written:

$$
\begin{aligned}
P_{11}&=(1-P_f)^2\, P^{II}_{11} = (1-P_f)^2(1-\alpha_1\theta_1)\\
P_{13}&=(1-P_f)^2\, P^{II}_{13} = (1-P_f)^2\varepsilon_2\,\alpha_1\,\theta_1\\
P_{14}&=(1-P_f)^2\, P^{II}_{14} = (1-P_f)^2(1-\varepsilon_2)\alpha_1\theta_1
\end{aligned}
\qquad (4.10)
$$

The transition to state $2\equiv(0\ 2)$ (or $6\equiv(2\ 0)$) could only take place if the system decided to go to state $1\equiv(0\ 1)$ (or $3\equiv(1\ 0)$) and there was a failure. The reason is that there is already a 0 in the state so the other element had to be a 1. Therefore

$$
\begin{aligned}
P_{12}&=P_4(1-P_f)\, P^{II}_{11} = P_f(1-P_f)\,(1-\alpha_1\theta_1)\\
P_{16}&=P_f(1-P_f)\, P^{II}_{13} = P_f(1-P_f)\,\varepsilon_2\alpha_1\theta_1
\end{aligned}
\qquad (4.11)
$$

The transition to state $5\equiv(1\ 2)$ or $7\equiv(2\ 1)$ can happen in two ways. Either the system decided to go to state $3\equiv(1\ 0)$ or $1\equiv(0\ 1)$ respectively and a failure brought computer 2 out of order or, the system intended to go to state $4\equiv(1\ 1)$ and the same failure happened thus we have

$$
\begin{aligned}
P_{15}&=P_f(1-P_f)(P^{II}_{13} + P^{II}_{14}) = P_f(1-P_f)\alpha_1\theta_1\\
P_{17}&=P_f(1-P_f)(P^{II}_{11} + P^{II}_{14}) = P_f(1-P_f)(1-\varepsilon_2\alpha_1\theta_1)
\end{aligned}
\qquad (4.12)
$$

Finally the transition to state $8\equiv(2\ 2)$ take place when there are two failures, no matter what transition was decided upon, hence

$$P_{18}=P_f^{\,2} \tag{4.13}$$

row 2: initial state 2≡(0 2)

From this state the automatic decision is "bring a copy from outside to computer 1". If there is no failure in computer 1 and computer 2 remains out of order, then the system goes to state 5.

Therefore

$$P_{2\ 5}=(1-P_f)(1-P_r) \tag{4.14}$$

If, on the other hand, computer 2 is restored and computer 1 does not fail the system will be in state 3 in the next instant of time and this event will occur with probability

$$P_{2\ 3}=(1-P_f)P_r \tag{4.15}$$

Considering now the case when computer 1 fails, different transitions appear

$$\begin{aligned} P_{2\ 6}&=P_fP_r \\ P_{2\ 8}&=P_f(1-P_r) \end{aligned} \tag{4.16}$$

row 3 and row 4 are obtained in the same way as row 1.

row 5: [initial state 5≡(1 2)] has the same transition probabilities as row 2, because as we said before, the decision of going from 2 to 5 is automatic, and then the possible transitions from 5 are only due to failures or recoveries in the computers(remember the sequence of events at the end of section IV-3 that decided a transition). Thus

$P_{5\ 5}=P_{2\ 5}=(1-P_f)(1-P_r)$

$P_{5\ 3}=P_{2\ 3}=(1-P_f)\ P_r$

$P_{5\ 6}=P_{2\ 6}=P_f\ P_r$

$P_{5\ 8}=P_{2\ 8}=P_f(1-P_f)$

<u>row 6</u>: $\left[\text{initial state } 6\equiv(2\ 0)\right]$ is obtained in the same way as row 2 but interchanging the computers.

$$\begin{aligned}&P_{6\ 1}=(1-P_f)P_r\\&P_{6\ 2}=P_fP_r\\&P_{6\ 7}=(1-P_f)(1-P_r)\\&P_{6\ 8}=P_f(1-P_r)\end{aligned}\qquad(4.17)$$

<u>row 7</u>: $\left[\text{initial state } 7\equiv(2\ 1)\right]$ is identical to row 6 for the same reason than row 5 was identical to row 2.

<u>row 8</u>: initial state $8\equiv(2\ 2)$

From this state there is no available decisions and the only solution is to wait. The probabilities are:

$$\begin{aligned}&P_{8\ 0}=P_r^{\ 2}\\&P_{8\ 2}=P_{8\ 6}=P_r(1-P_r)\\&P_{8\ 8}=(1-P_r)^2\end{aligned}\qquad(4.18)$$

The vector dynamic equation is

$$Y(t+1)=P(t,u)Y(t)$$

where $P(t,u)$ is the transition probability matrix.

It can be checked that the transition probability matrix obtained above, is a well defined stochastic matrix in the sense that all rows add up to one. In the next section we will see that these results easily generalize to any value of NC and that the transition probabilities can be obtained easily and efficiently. Then in the next chapter these results will be applied to a numerical example.

To conclude this section we point out that in the case of a network with failures, the terms due to updating traffic are not only function of the state to go but also of the present state . The reason is very simple: a computer that is not working cannot generate updating traffic, furthermore if the state is such that no copies are present in the system, ((0 2) for instance) no updating traffic can be generated either. A flow-chart showing how to compute the updating traffic for any network is presented in the next section.

IV.4. Recursive Equations for NC=3 and a general NC considering node Failures

In this section the results of section IV-3 will be extended to any NC and general rules showing how to obtain the per unit-time-costs, updating traffic costs, and transition probabilities will be developed.

First the case NC=3 will be examined. The number of states is $M=3^3=27$, and they are all ternary numbers from m=0 (0≡(0 0 0)) to m=26 (26≡(2 2 2)). Let us begin deriving the per unit time and terminal costs. We have the following cases:

a) the state is such that no copies are present and at least one computer is working

$$\lambda_m=\min\{C_{0j}\} \quad j\in\{\text{set of working computers}\} \tag{4.20a}$$

b) all computers are out of work

$$\lambda_M = 0 \tag{4.20b}$$

c) general case

$$\lambda_m= \sum_{i\epsilon I} C_i+ \sum_{j\epsilon J} \theta_j(t) \min_{i\epsilon I} \{C_{ij}\} \tag{4.20c}$$

where I is the set of working computers with a copy, and J is the set of working computers without a copy.

Some of these costs are obtained bellow as illustration (it is assumed $C_{01} < C_{02} < C_{03}$)

$$\begin{array}{ll}
0\equiv(0\ 0\ 0) & \lambda_0(t) = C_{01} \\
4\equiv(0\ 1\ 1) & \lambda_4(t) = C_2+C_3+\theta_1\min(C_{21},C_{31}) \\
7\equiv(0\ 2\ 1) & \lambda_7(t)=C_3+\theta_1 C_{31} \\
18\equiv(2\ 0\ 0) & \lambda_{18}(t)=C_{02} \\
24\equiv(2\ 2\ 0) & \lambda_{24}(t)=C_{03} \\
26\equiv(2\ 2\ 2) & \lambda_{26}(t)=0
\end{array} \tag{4.21}$$

The terminal costs are obtained in the same way except that as discussed before, there will be a zero in the place where there was a C_{0i}. A flow-chart showing how to obtain terminal and per-unit time cost for any value NC is sketched in Fig. IV-3.

The way to compute the cost due to updating traffic for all possible transitions in a network with NC computers is shown in the flow-chart of Fig. IV.2. We call R(I,F) to the (I,F)th element of the updating traffic matrix.

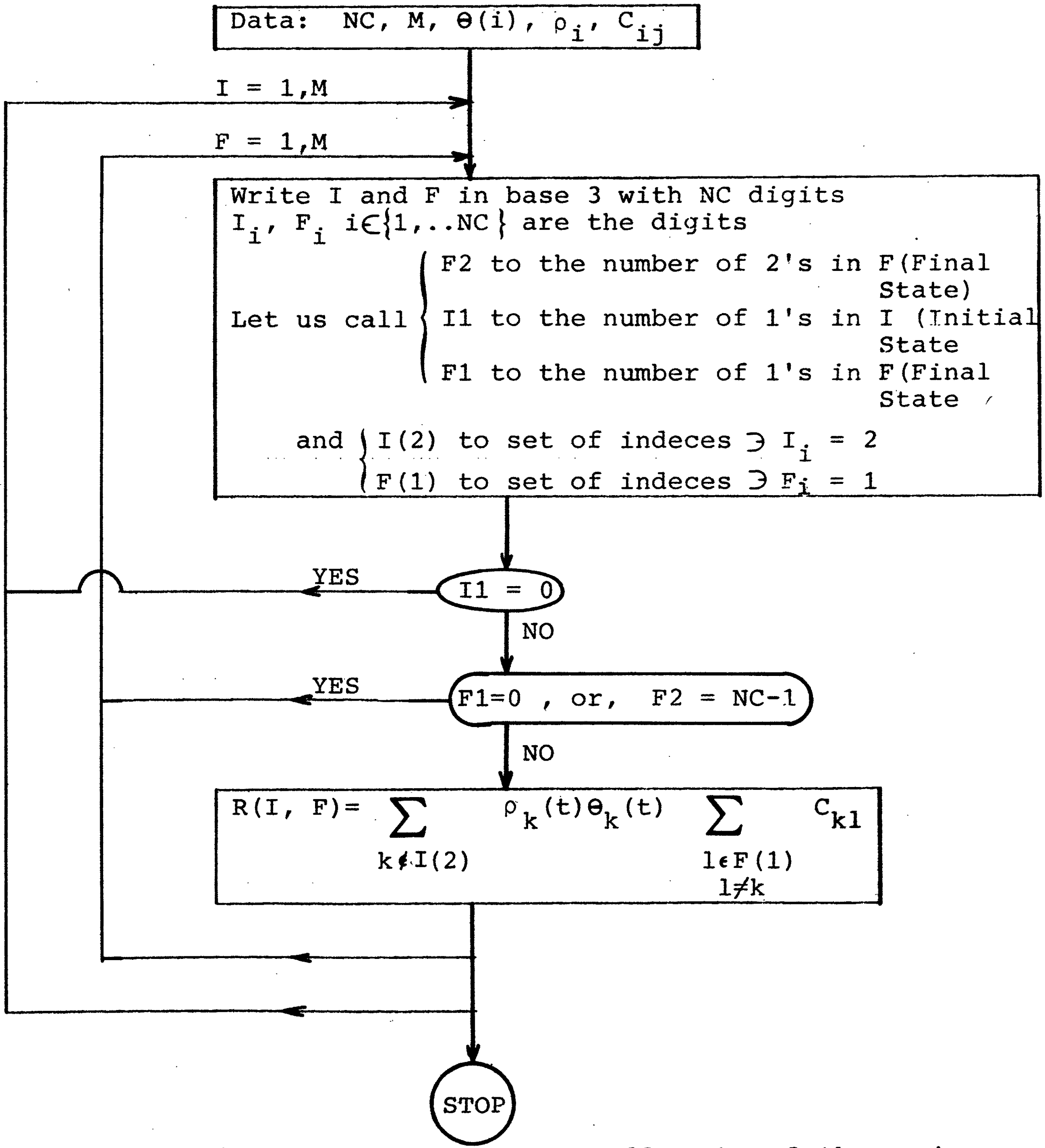

Data: NC, M, $\Theta(i)$, ρ_i, C_{ij}
I = 1,M
F = 1,M
Write I and F in base 3 with NC digits
I_i, F_i $i \in \{1,..NC\}$ are the digits
Let us call
F2 to the number of 2's in F(Final State)
I1 to the number of 1's in I (Initial State
F1 to the number of 1's in F(Final State
and
I(2) to set of indeces $\ni I_i = 2$
F(1) to set of indeces $\ni F_i = 1$
YES
I1 = 0
NO
YES
F1=0 , or, F2 = NC-1
NO
$R(I, F) = \sum_{k \notin I(2)} \rho_k(t)\Theta_k(t) \sum_{\substack{l \in F(1) \\ l \neq k}} C_{kl}$
STOP

Fig. IV-2. Flow-chart of updating traffic when failures in the computer are considered

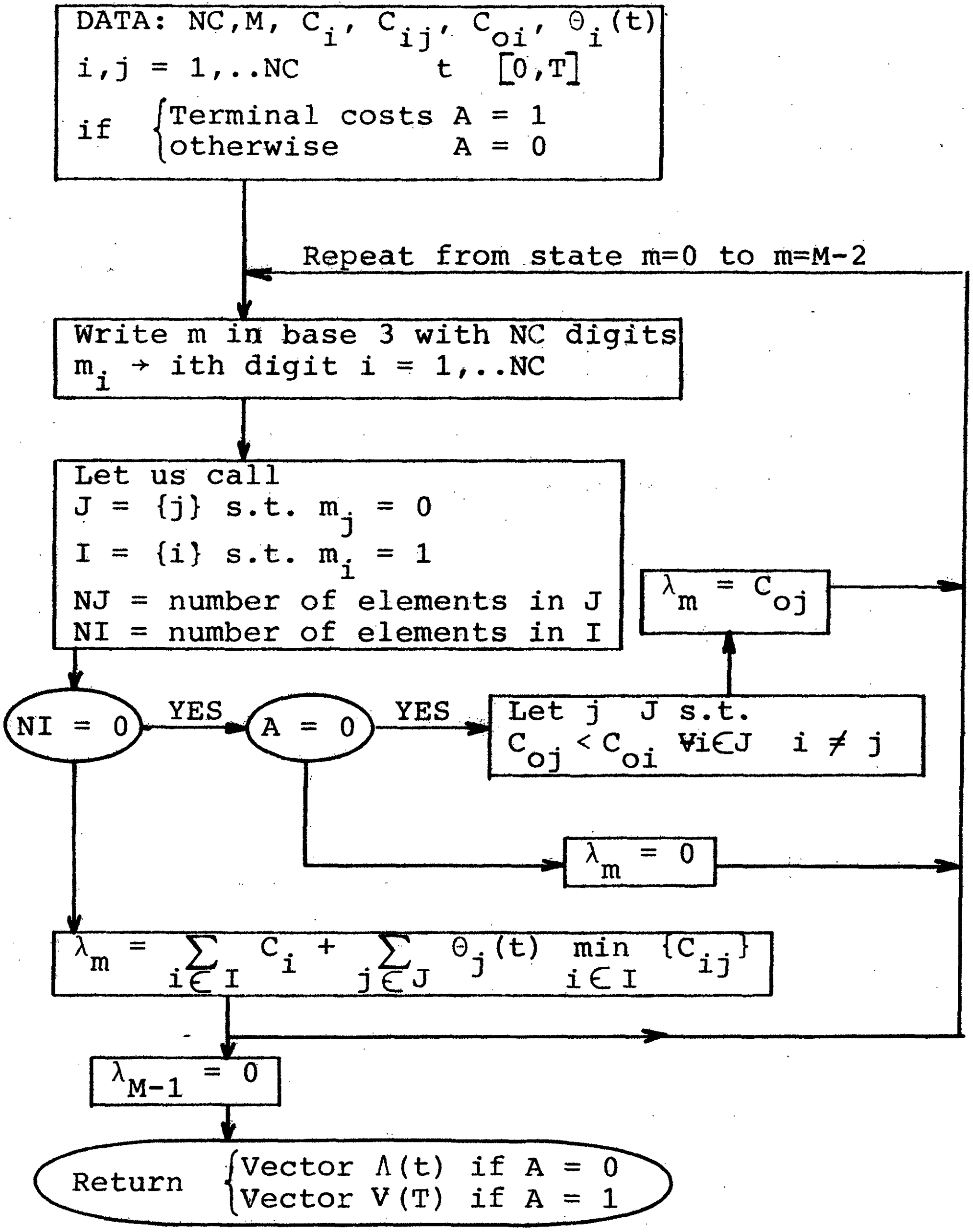

DATA: NC,M, C_i, C_{ij}, C_{oi}, $\Theta_i(t)$
i,j = 1,..NC t [0,T]
if {Terminal costs A = 1
otherwise A = 0
Repeat from state m=0 to m=M-2
Write m in base 3 with NC digits
m_i → ith digit i = 1,..NC
Let us call
J = {j} s.t. m_j = 0
I = {i} s.t. m_i = 1
NJ = number of elements in J
NI = number of elements in I
NI = 0
YES
A = 0
YES
Let j J s.t.
$C_{oj} < C_{oi}$ ∀i∈J i ≠ j
$\lambda_m = C_{oj}$
$\lambda_m = 0$
$\lambda_m = \sum_{i\in I} C_i + \sum_{j\in J} \Theta_j(t) \min_{i\in I} \{C_{ij}\}$
$\lambda_{M-1} = 0$
Return {Vector $\Lambda(t)$ if A = 0
Vector V(T) if A = 1

Fig. IV-3. Flow-Chart showing how to obtain terminal and per-unit-time costs

In fig. IV-4 we find the tableau of possible transitions for the case NC=3 with symbols as defined in section IV-3. The basic rules for the formation of the tableau are the same as for the case NC=2;

1) No transition from a state component 2 to a state component 1 is possible
2) If there are only 0's and 2's in the state vector the only possible transition is to bring a copy to the computer with smallest C_{0i}

Example of impossible transitions

$$\begin{array}{llll} \text{from} & 0\ 2\ 2 & \text{to} & 0\ 2\ \textcircled{1} \\ \text{from} & 0\ 0\ 0 & \text{to} & 1\ 2\ \textcircled{1} \\ \text{from} & 0\ 2\ 2 & \text{to} & \textcircled{0}\ 2\ 0 \\ \text{from} & 1\ 2\ 2 & \text{to} & \textcircled{0}\ 2\ \textcircled{1} \end{array} \qquad (4.22)$$

where the elements causing trouble have been circled out.

To obtain the general rules for the transition probabilities let us begin analyzing some examples.

We define, as before, P^L_{ij} as the transition probability from i to j in a network with L computers when no failures are considered

- The transition due to failures or recoveries only are obtained with the same basic rules as for the case NC=2, for instance

$$\begin{array}{l} 0=(0\ 0\ 0) \longrightarrow 9=(1\ 0\ 0), P_{0\ 9}=(1-P_f)^3 \\ 6=(0\ 2\ 0) \longrightarrow 17=(1\ 2\ 2), P_{0\ 17}=(1-P_f)(1-P_r)P_f \\ 25=(2\ 2\ 1) \longrightarrow 2=(0\ 0\ 2), P_{25\ 2}=P_r^2P_f \end{array} \qquad (4.23)$$

																												State Vector		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	1	2	3
0										*		X				X		X	X		X				X		X	0	0	0
1		⊠	X	⊠	⊠	X	X	X	X	⊠	⊠	X	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	1
2										X		*				X		X	X		X				X		X	0	0	2
3		⊠	X	⊠	⊠	X	X	X	X	⊠	⊠	X	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	0
4		⊠	X	⊠	⊠	X	X	X	X	⊠	⊠	X	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1
5				X		⊠	X		X	X		⊠	X		⊠	X		X	X		X	X		X	X		X	0	1	2
6										X		X				*		X	X		X				X		X	0	2	0
7		X	X					⊠	X	X	X	X				⊠	⊠	X	X	X	X				X	X	X	0	2	1
8										X		X				X		*	X		X				X		X	0	2	2
9		⊠	X	⊠	⊠	X	X	X	X	⊠	⊠	X	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	1	0	0
10		⊠	X	⊠	⊠	X	X	X	X	⊠	⊠	X	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	1	0	1
11				X		⊠	X		X	X		⊠	X		⊠	X		X	X		X	X		X	X		X	1	0	2
12		⊠	X	⊠	⊠	X	X	X	X	⊠	⊠	X	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	0
13		⊠	X	⊠	⊠	X	X	X	X	⊠	⊠	X	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1
14				X		⊠	X		X	X		⊠	X		⊠	X		X	X		X	X		X	X		X	1	1	2
15		X	X					⊠	X	X	X	X				⊠	⊠	X	X	X	X				X	X	X	1	2	0
16		X	X					⊠	X	X	X	X				⊠	⊠	X	X	X	X				X	X	X	1	2	1
17										X		X				X		*	X		X				X		X	1	2	2
18				X		X	X		X													*		X	X		X	2	0	0
19		X	X	X	X	X	X	X	X											⊠	X	⊠	⊠	X	X	X	X	2	0	1
20				X		X	X		X													X		*	X		X	2	0	2
21		X	X	X	X	X	X	X	X											⊠	X	⊠	⊠	X	X	X	X	2	1	0
22		X	X	X	X	X	X	X	X											⊠	X	⊠	⊠	X	X	X	X	2	1	1
23				X		X	X		X													X		*	X		X	2	1	2
24		X	X					X	X											X	X					*	X	2	2	0
25		X	X					X	X											X	X					*	X	2	2	1
26	X		X				X		X										X		X				X		X	2	2	2

Fig. IV-4. Tableau of possible transitions for N C=3.

- The transitions from states with no inoperative computers are also directed by the same basic rules as the corresponding transitions for NC=2, see section IV-3, thus from state 1=(0 0 1) to state 10=(1 0 1)

$$P_{1\ 10}=(1-P_f)^3\ P^{III}_{1\ 10} \qquad (4.24)$$

from state 1=(0 0 1) to state 5=(0 1 2)

$$P_{1\ 5}=(1-P_f)^2 P_f (P^{III}_{1\ 3} + P^{III}_{1\ 4})$$

where 3 and 4 are the states obtained from 5 replacing the 2 by 0 and 1, that is

5 = (0 1 2)

3 = (0 1 0)

4 = (0 1 1)

- If the transition is to a state with two 2's then the number of $P^L_{i\ j}$'s contributing to the probability is larger (3 or 4 depending on the case)

from state 1=(0 0 1) to 17 = (1 2 2)

$$P_{1\ 17}=(1-P_f)^2 P_f (P^{III}_{1\ 9} + P^{III}_{1\ 10} + P^{III}_{1\ 12} + P^{III}_{1\ 13}) \qquad (4.25)$$

where as before 9, 10, 12 and 13 are the states obtained from 17 replacing the 2's by all possible combinations of 0's and 1's, that is

$$\text{from } 17 = (1\ 2\ 2) \text{ we obtain } \begin{cases} (1\ 0\ 0) = 9 \\ (1\ 0\ 1) = 10 \\ (1\ 1\ 0) = 12 \\ (1\ 1\ 1) = 13 \end{cases}$$

- If the transition is from state 1=(0 0 1) to state 24=(2 2 0) then

  $$P_{1\ 24}=(1-P_f)^2P_f(P^{III}_{1\ 3} + P^{III}_{1\ 9} + P^{III}_{1\ 12}) \qquad (4.26)$$

  where now we only have three terms in the sum because state 0=(0 0 0) was unallowed for the case of no failures
- The probability transitions from states with only two computers in working condition will be formed up from terms $P^{II}_{i\ j}$. The subscripts i and j will be obtained from a modified 2-components states resulting from the previous one after the elimination of the element that was equal to 2 in the initial states and applying previous rules. That is from state 7=(0 2 1) the modified initial state will be (0 1)= 1, therefore a transition from 7 to 11=(1 0 2) will have the following probability

  $$P_{7\ 11}=(1-P_f)P_rP_f(P^{II}_{1\ 2} + P^{II}_{1\ 3}) \qquad (4.27)$$

  where 2 and 3 are obtained from 7 and 11 as follows

  $$\begin{array}{l} 7=(0\textcircled{2}1) \\ 11=(1\textcircled{0}2) \longrightarrow \begin{cases} (1\ 0)\ =\ 2 \\ (1\ 1)\ =\ 3 \end{cases} \end{array}$$

The rest of transition probabilities can be obtained in the same way. It can be easily checked again that the resulting transition matrix has all the properties needed by a stochastic matrix. The dynamic equation for the state is, as before,

$$Y(t+1) = P(t,u)\ Y(t)$$

These results generalize to a network with any number of computers. It will nevertheless be convenient to sketch in a flow-chart the ordered set of steps we have to take to obtain the element of the transition matrix. To simplify the flow chart, let us suppose that we have the following subroutines available (see Appendix C)

BASE 3 (NC,N,NB3,NO,N1,N2) → Write N in base 3 with NC components
($3^{NC} \geq N$)
NB3(I) is the Ith component
N0 = number of 0's in NB3
N1 = number of 1's in NB3
N2 = number of 2's in NB3

BITS(N,NC,NB2) —— Write N in base 2 with NC digits
NB2(I) is the Ith component

ROWPRO (NC,ν,λ,Θ,Π,IND) → Obtain the νth row of the transition probability of a network with NC components and no failures, when the rates are

$\Theta(i)$ i=1,NC and

the decision is "go to state λ" . This result is obtained with the flow-chart of Fig.III-3.
$\Pi(J)$ is the Jth element of this row
IND is an output index s.t.

$$\text{IND}\begin{cases}1 \text{ if some} \Pi(J)=1 \quad (*)\\ 0 \text{ otherwise}\end{cases}$$

we will now call

N - the row (or present state) being computed

......

(*) Notice that if this happens the only nonzero element in the row is $\Pi(\lambda)=1$

M = the number of states $M=3^N$

NC - the number of computers in the network under consideration

M' = the number of states for the auxiliar subsystem without failures

R(J,I) - the updating traffic cost generated in a transition from I to J and obtained with the flow-chart of Fig. IV-2

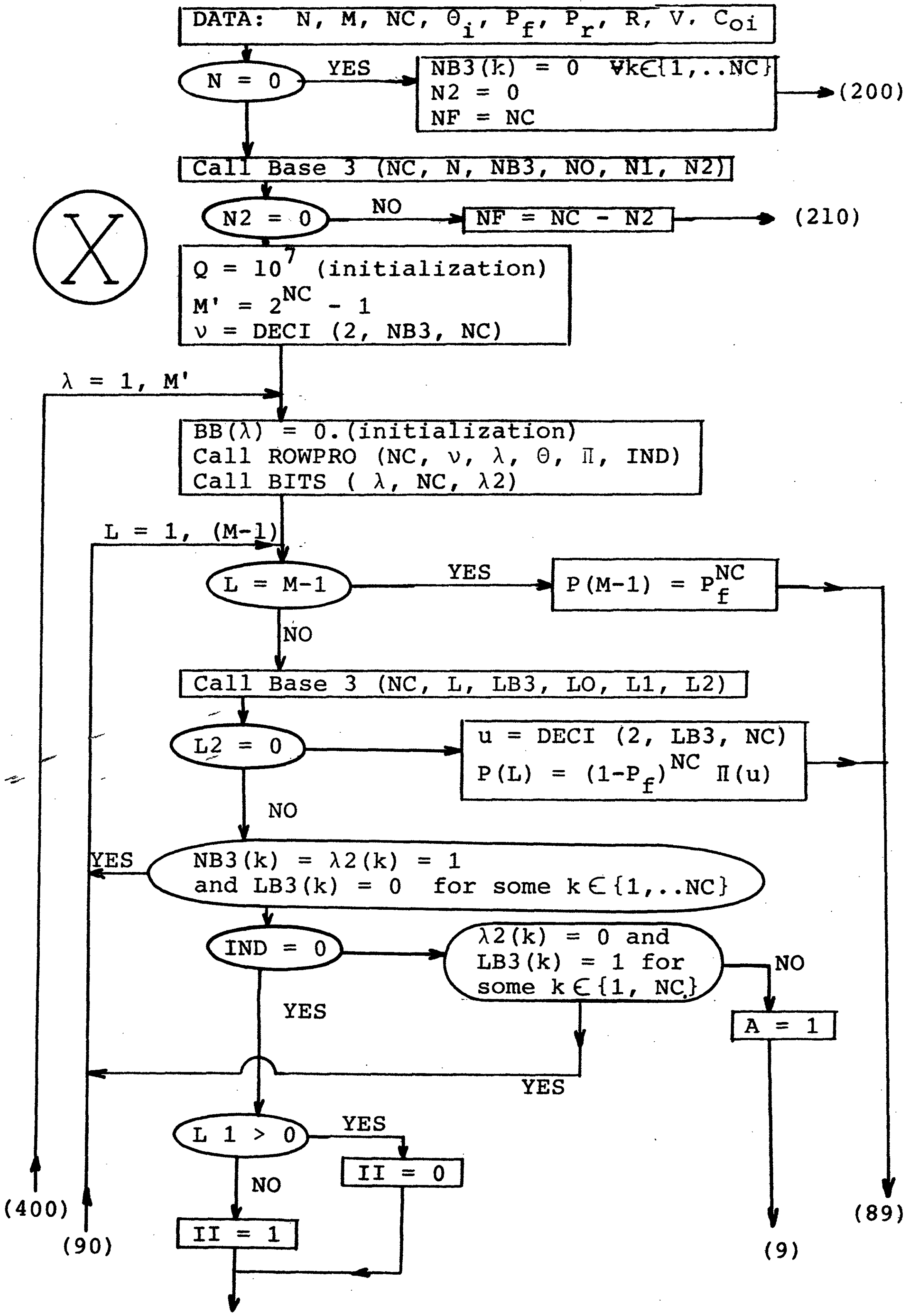
R_N - the Nth column of the matrix $\underline{R} \equiv \{R(J,I)\}$

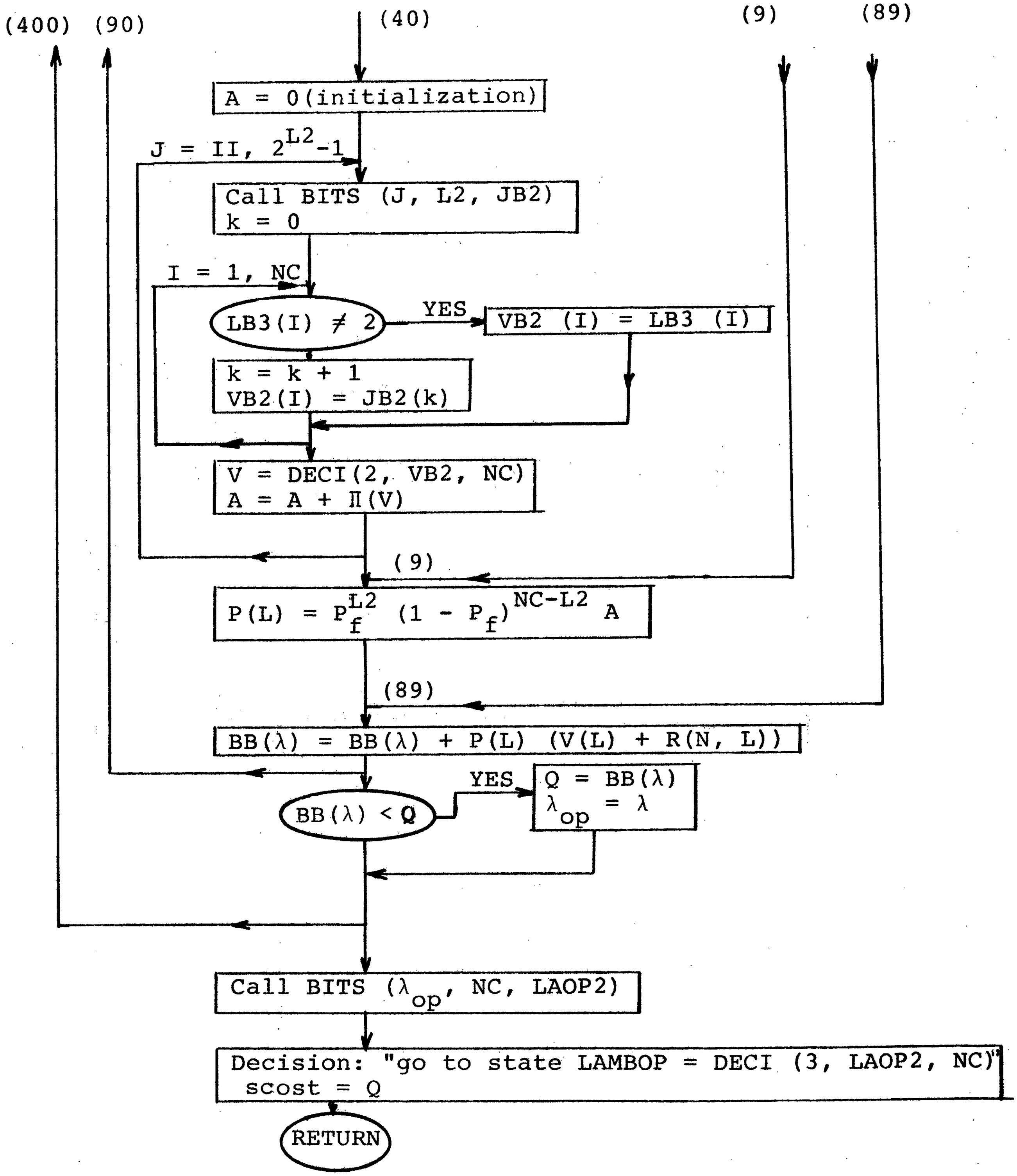
For simplicity of representation and convenience of computation the algebraic representation of the entries will not be derived. Instead we present an algorithm to calculate their numerical values after deciding upon the control variables. Having the transition probabilities the optimization process can be carried out directly by moving the decision to all its possible values, performing the product with the matrix (V + R) and choosing the decision giving the smallest value. This is what is done in the flow-chart. The variable λ will control the admissible decisions and the variable L the corresponding element within the row. The results after the computation will be

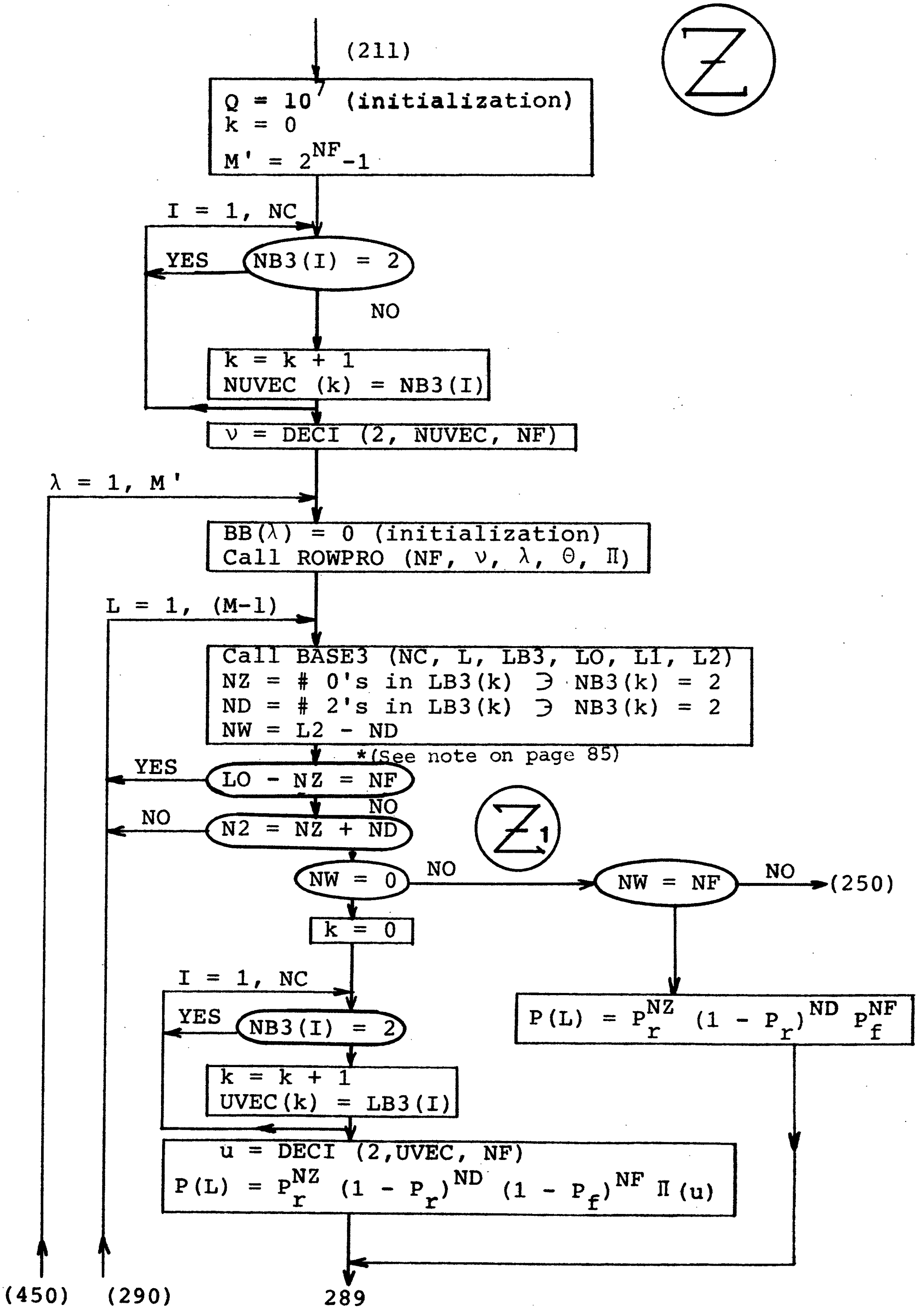
"Go to state LAMBOP" ——— actual optimal decision from state N

scost ——— actual cost due to this transition resulting from the product → row N x $(V + R_N)$

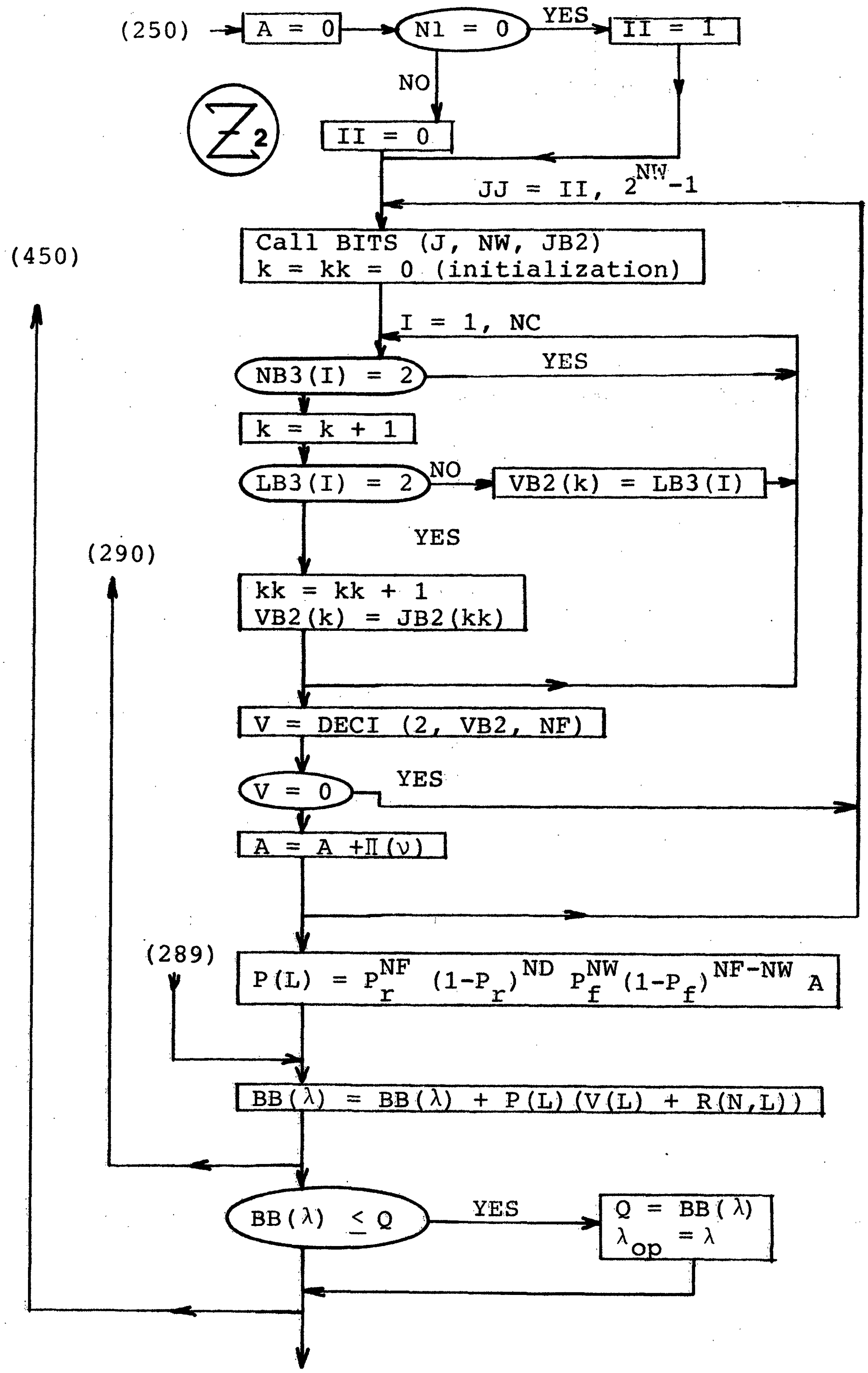
Looking carefully at the flow-chart it can be seen that the first thing it does is an assignment of state N to one of the three basic types of states it considers. In this way the rest of the computation will take place in one of the three main branches X, Y or Z.

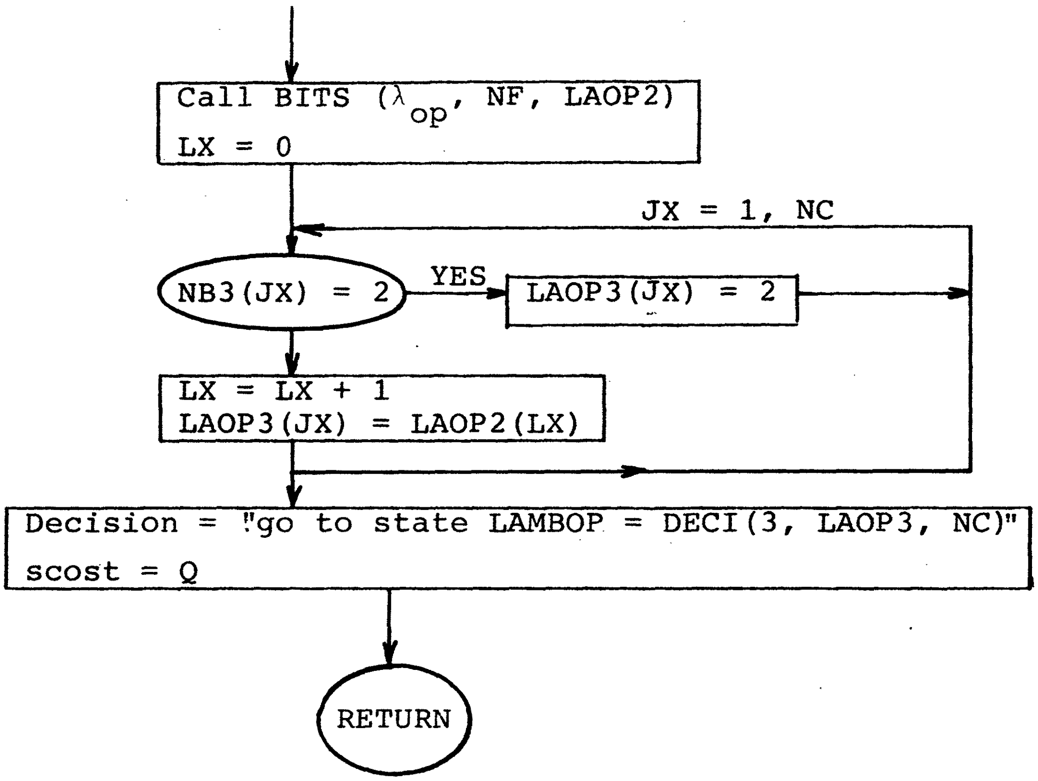

DATA: N, M, NC, Θ_i, P_f, P_r, R, V, C_{oi}
N = 0
YES
NB3(k) = 0 ∀k∈{1,..NC}
N2 = 0
NF = NC
(200)
Call Base 3 (NC, N, NB3, NO, N1, N2)
X
N2 = 0
NO
NF = NC - N2
(210)
Q = 10^7 (initialization)
M' = 2^{NC} - 1
ν = DECI (2, NB3, NC)
λ = 1, M'
BB(λ) = 0.(initialization)
Call ROWPRO (NC, ν, λ, Θ, Π, IND)
Call BITS (λ, NC, λ2)
L = 1, (M-1)
L = M-1
YES
P(M-1) = P_f^{NC}
NO
Call Base 3 (NC, L, LB3, LO, L1, L2)
L2 = 0
u = DECI (2, LB3, NC)
P(L) = $(1-P_f)^{NC}$ Π(u)
NO
YES
NB3(k) = λ2(k) = 1
and LB3(k) = 0 for some k∈{1,..NC}
IND = 0
λ2(k) = 0 and
LB3(k) = 1 for
some k∈{1, NC}
NO
YES
A = 1
YES
L 1 > 0
YES
II = 0
NO
II = 1
(400)
(90)
(9)
(89)


(400)
(90)
(40)
(9)
(89)
A = 0(initialization)
J = II, 2^{L2}-1
Call BITS (J, L2, JB2)
k = 0
I = 1, NC
LB3(I) ≠ 2
YES
VB2 (I) = LB3 (I)
k = k + 1
VB2(I) = JB2(k)
V = DECI(2, VB2, NC)
A = A + Π(V)
(9)
$P(L) = P_f^{L2} (1 - P_f)^{NC-L2} A$
(89)
BB(λ) = BB(λ) + P(L) (V(L) + R(N, L))
BB(λ) < Q
YES
Q = BB(λ)
λ_{op} = λ
Call BITS (λ_{op}, NC, LAOP2)
Decision: "go to state LAMBOP = DECI (3, LAOP2, NC)"
scost = Q
RETURN

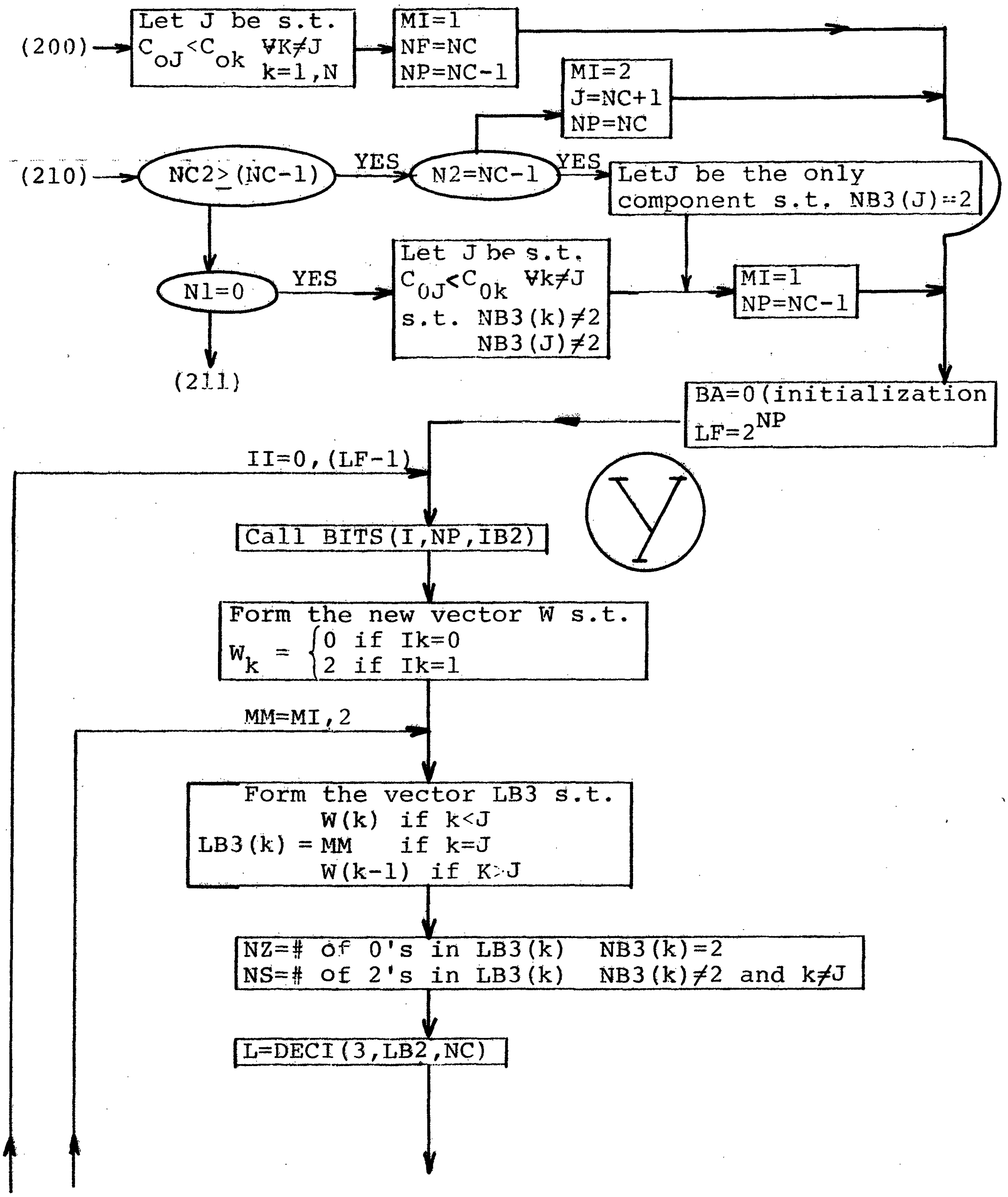

(211)
Z
Q = 10^7 (initialization)
k = 0
M' = 2^NF -1
I = 1, NC
YES
NB3(I) = 2
NO
k = k + 1
NUVEC (k) = NB3(I)
ν = DECI (2, NUVEC, NF)
λ = 1, M'
BB(λ) = 0 (initialization)
Call ROWPRO (NF, ν, λ, Θ, Π)
L = 1, (M-1)
Call BASE3 (NC, L, LB3, LO, L1, L2)
NZ = # 0's in LB3(k) ∋ NB3(k) = 2
ND = # 2's in LB3(k) ∋ NB3(k) = 2
NW = L2 - ND
*(See note on page 85)
YES
LO - NZ = NF
NO
NO
N2 = NZ + ND
Z1
NW = 0
NO
NW = NF
NO
(250)
k = 0
I = 1, NC
YES
NB3(I) = 2
k = k + 1
UVEC(k) = LB3(I)
u = DECI (2,UVEC, NF)
P(L) = P_r^NZ (1 - P_r)^ND (1 - P_f)^NF Π(u)
P(L) = P_r^NZ (1 - P_r)^ND P_f^NF
(450)
(290)
289

*(see note on page 85)





*See Fortran listing in Appendix C for further zero transition probability checkings at this point.


(200)
Let J be s.t.
$C_{oJ}<C_{ok}$ ∀K≠J
k=1,N
MI=1
NF=NC
NP=NC-1
MI=2
J=NC+1
NP=NC
(210)
NC2≥(NC-1)
YES
N2=NC-1
YES
LetJ be the only
component s.t. NB3(J)=2
N1=0
YES
Let J be s.t.
$C_{0J}<C_{0k}$ ∀k≠J
s.t. NB3(k)≠2
NB3(J)≠2
MI=1
NP=NC-1
(211)
BA=0(initialization
$LF=2^{NP}$
II=0,(LF-1)
Y
Call BITS(I,NP,IB2)
Form the new vector W s.t.
$W_k = \begin{cases} 0 \text{ if } Ik=0 \\ 2 \text{ if } Ik=1 \end{cases}$
MM=MI,2
Form the vector LB3 s.t.
LB3(k) = W(k) if k<J
MM if k=J
W(k-1) if K>J
NZ=# of 0's in LB3(k) NB3(k)=2
NS=# of 2's in LB3(k) NB3(k)≠2 and k≠J
L=DECI(3,LB2,NC)

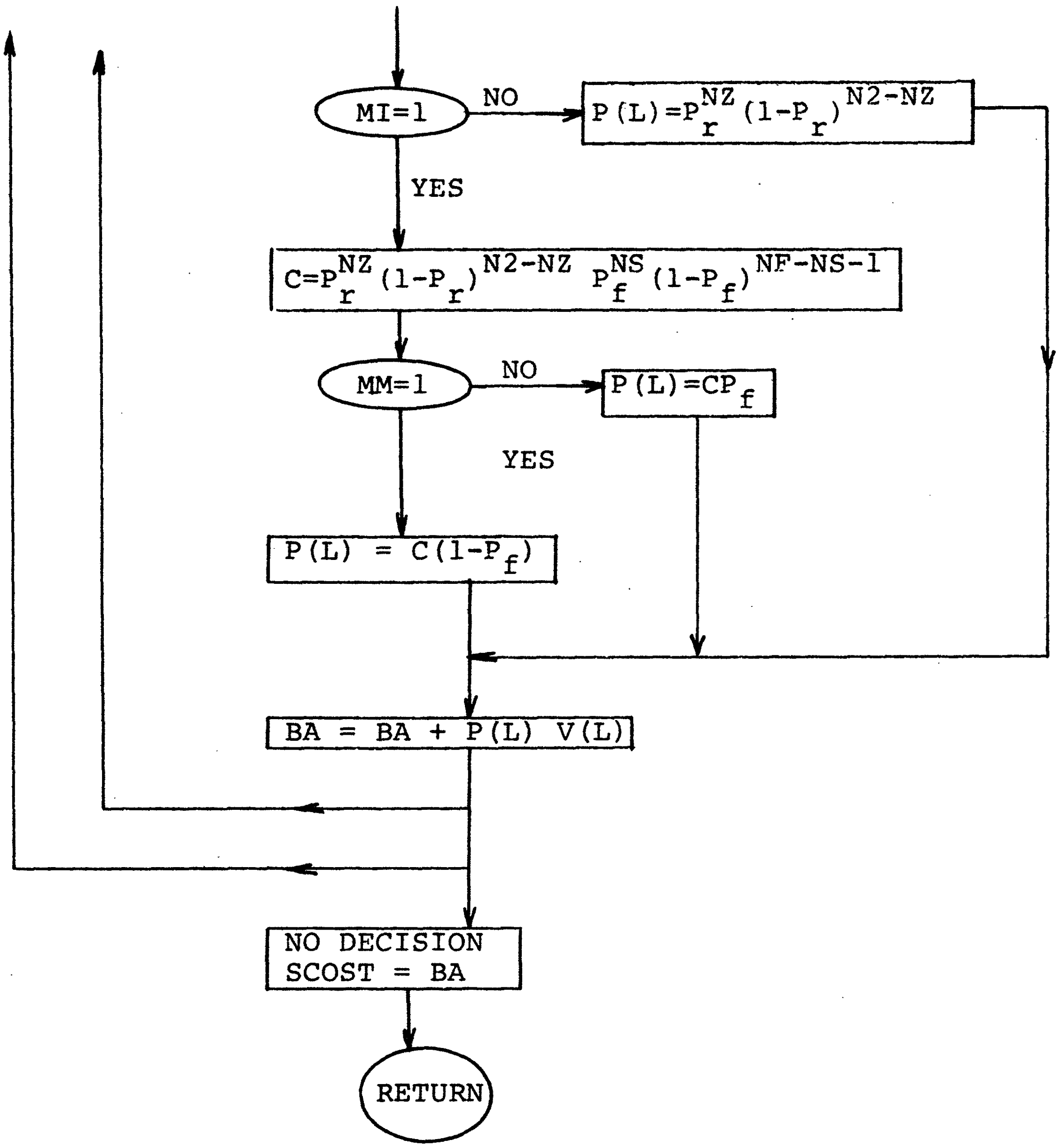


Fig. IV.5. Flow-chart showing how to obtain the transition matrix and the optimal decision for a network with failures.

Branch X corresponds to states with no 2's amongtheir components

Branch Y corresponds to states where no decision is available

Branch Z to the rest of states. This branch has a subdivision depending on the destination state
Z_1 if the destination state is such that to reach it we have only one possibility
Z_2 otherwise

In branches X and Z every possible transition (index L) from N is tested and only the transitions that give a nonzero probability are obtained. In branch Y, only the nonzero probability transitions are generated. Functioning in this way the algorithm is very efficient because only the nonzero terms (∿30% of the total) are obtained and there is no waisted time performing zero computation. The algorithm was programmed in Fortran, using Assembler for BITS Subroutine (see Appendix C)

Now that the transition matrix and optimal decisions have been obtained the rest of the process is identical to the case with no failures. (Fig. III-4). The whole process is represented in Fig. IV-5 and the matrix recursive eq. will be

$$V^*(t) = \Lambda(t)+P(t,u^*)\ V^*(t+1)+P(t,u^*)\boxtimes R(t) \qquad (4.29)$$

where R(t) is now an M by M matrix and $\boxtimes$ means that we make the scalar product of the ith row of P(.,.) with the ith column of R(:) to obtain the ith component of a column vector.

Let us briefly analyze how the actual dynamic behaviour of a network, with present characteristics, will be, when the

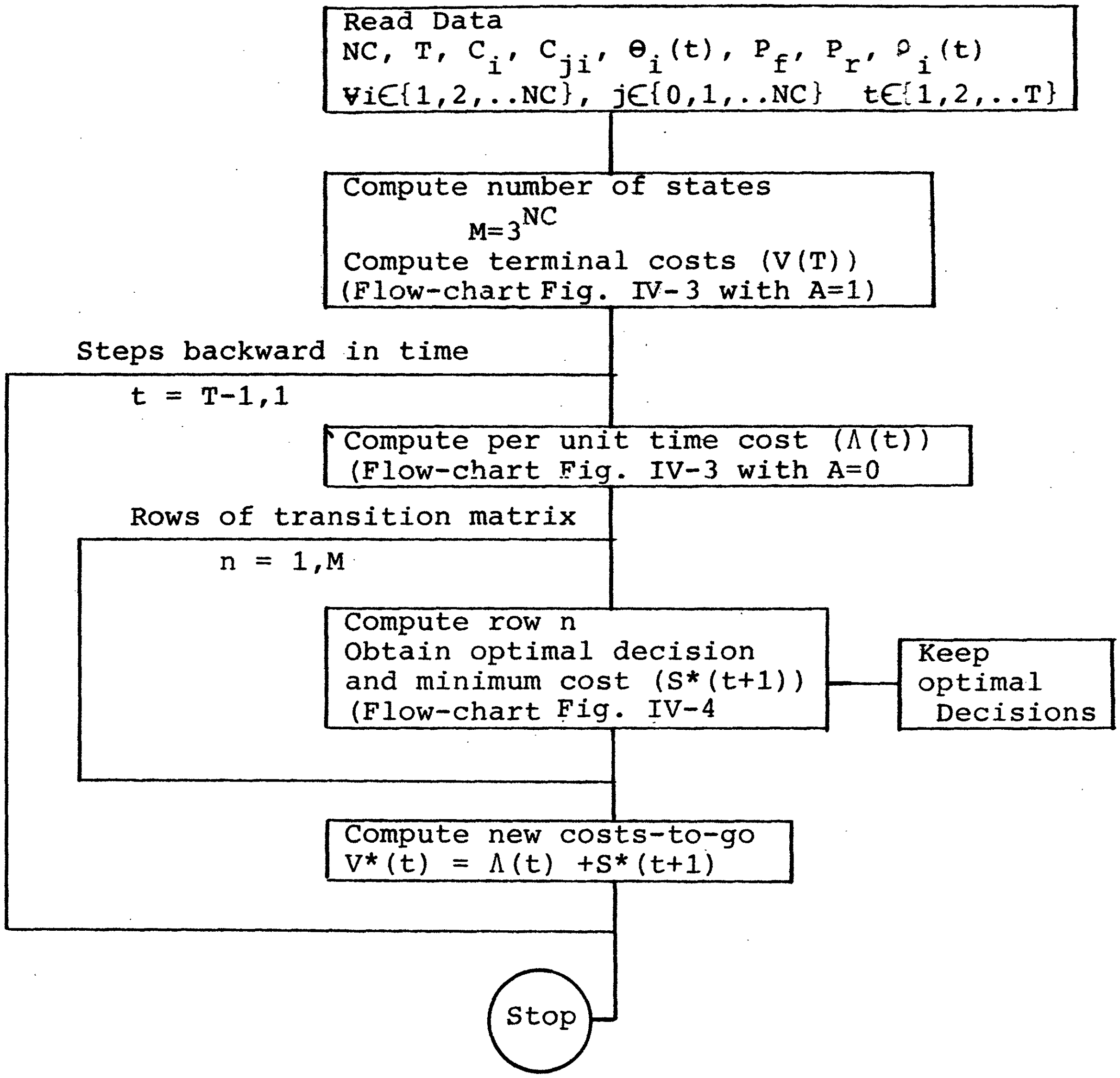

Read Data
NC, T, C_i, C_{ji}, $\theta_i(t)$, P_f, P_r, $\rho_i(t)$
$\forall i \in \{1,2,..NC\}$, $j \in \{0,1,..NC\}$ $t \in \{1,2,..T\}$
Compute number of states
$M=3^{NC}$
Compute terminal costs (V(T))
(Flow-chart Fig. IV-3 with A=1)
Steps backward in time
t = T-1,1
Compute per unit time cost (Λ(t))
(Flow-chart Fig. IV-3 with A=0
Rows of transition matrix
n = 1,M
Compute row n
Obtain optimal decision
and minimum cost (S*(t+1))
(Flow-chart Fig. IV-4
Keep
optimal
Decisions
Compute new costs-to-go
V*(t) = Λ(t) +S*(t+1)
Stop

Fig. IV-6. Optimization Process in Network with Failures

sequence of optimal decisions is applied. Let us suppose that the probability of recovery for a computer that is not working is not equal to 1, or in other words that the recovery is not instantaneous. If we assume that the process begins with all the computers working, the system will begin to make transitions among states with no 2's among their components. Once a failure takes place the system will change its "state space" to states that have a 2 in the position of the failure; it will remain making transition in this new "state space" until one of two possible events will take place: either there is another failure (or more), or the failed computer begins to work. The process will continue in this way. We see therefore that the whole state space can be divided into various subspaces such that the system will remain most of its time making transitions in those subspaces and eventually will move from one subspace to another. There will be as many subspaces as different vectors we can form with NC components and two symbols (one for 2 and the other for 1 or 0). All the vectors with same frame of 2 components (and at least one 1) will belong to the same subspace.

For NC=3 some of the subspaces will be:

$$S_1 \begin{cases} 0\ 0\ 1 - 1 \\ 0\ 1\ 0 - 3 \\ 0\ 1\ 1 - 4 \\ 1\ 0\ 0 - 9 \\ 1\ 0\ 1 - 10 \\ 1\ 1\ 0 - 12 \\ 1\ 1\ 1 - 13 \end{cases} \qquad S_2 \begin{cases} 0\ 1\ 2 \\ 1\ 0\ 2 \\ 1\ 1\ 2 \end{cases} \qquad S_3 \begin{cases} 0\ 2\ 1 \\ 1\ 2\ 0 \\ 1\ 2\ 1 \end{cases} \qquad (4.30)$$

This particular behavior and subspaces division can also be illustrated if the transition tableau of Fig. IV-4 is written with a different state order as in Fig. IV-6 is shown. The system will be most of the time within the marked regions of the tableau and eventually will move from one region to another. The states with NC-1 elements equal to 2 (as(0 2 2)= 8) may be thoughts as degenerated subspaces or just subspaces with only one component.

	0	1	3	4	9	10	12	13	2	5	11	14	6	7	15	16	18	19	21	22	8	17	20	23	24	25	26
0					*						X				X		X					X	X		X		X
1		⊠	⊠	⊠	⊠	⊠	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3		⊠	⊠	⊠	⊠	⊠	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
4		⊠	⊠	⊠	⊠	⊠	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
9		⊠	⊠	⊠	⊠	⊠	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
10		⊠	⊠	⊠	⊠	⊠	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
12		⊠	⊠	⊠	⊠	⊠	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
13		⊠	⊠	⊠	⊠	⊠	⊠	⊠	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
2					X						*				X		X					X	X		X		X
5			X		X		X			⊠	⊠	⊠	X		X		X		X		X	X	X	X	X		X
11			X		X		X			⊠	⊠	⊠	X		X		X		X		X	X	X	X	X		X
14			X		X		X			⊠	⊠	⊠	X		X		X		X		X	X	X	X	X		X
6					X						X				*		X					X	X		X		X
7		X			X	X			X		X			⊠	⊠	⊠	X	X			X	X	X		X	X	X
15		X			X	X			X		X			⊠	⊠	⊠	X	X			X	X	X		X	X	X
16		X			X	X			X		X			⊠	⊠	⊠	X	X			X	X	X		X	X	X
18			X							X			X						*		X			X	X		X
19		X	X	X					X	X			X	X				⊠	⊠	⊠	X		X	X	X	X	X
21		X	X	X					X	X			X	X				⊠	⊠	⊠	X		X	X	X	X	X
22		X	X	X					X	X			X	X				⊠	⊠	⊠	X		X	X	X	X	X
8					X						X				X		X					*	X		X		X
17					X						X				X		X					*	X		X		X
20			X							X			X						X		X			*	X		X
23			X							X			X						X		X			*	X		X
29		X							X					X				X			X		X			*	X
25		X							X					X				X			X		X			*	X
26	X								X				X				X				X		X		X		X

Fig. IV.7. Reordered Transitions Tableau showing the "Subspace behavior" for NC=3.

CHAPTER V

NUMERICAL APPLICATIONS AND OTHER ANALYTICAL RESULTS

V.1 Time varying rates,no failures, no updating traffic.

Let us analyze the case of a network with two computers sharing a file according to time varying rates. We will apply the model of chapter III and we want to know the dynamic evolution of the states (i.e. allocation of the file) in order to minimize the total expected cost. We also want to study the evolution of the state dynamics and total cost as the storage cost varies from 0 to the value of the transmission cost. The transmission costs will all be taken to be equal to 1. The problem will be solved for storage costs equal to 0, 0.25, 0.5, 0.75, 1. The system will operate for a period of 20 time units, $[1,20]$. The rates are represented in Fig. V.1, and the results inTable V.1. In the columns called "evolution of states" we write the optimal decision ("go to state...") for every possible initial state and every instant of time.

Examining the table we can see that for the case of storage cost equal to zero the optimal decision is to always keep a copy of the file in each computer. The optimum initial state is state 3 and the optimal decision being at state 3 is always "remain in 3". This is the logical result because there is no payoff for keeping a copy in any computer. At t=19 we will leave the system at state 2; this decision is

	RATES		$C_{ij}= 1 \quad \forall i,j$	OPTIMAL DECISIONS EVOLUTION OF STATES				
TIME	COMPUTER		STORAGE COST	0.0	.25	.50	.75	1.0
	1	2	PRESENT STATE	123	123	123	123	123
20	0.8	0.0						
19	0.8	0.4		222	222	222	222	222
18	0.8	0.8		333	333	222	222	222
17	0.8	0.8		333	333	333	333	222
16	0.0	0.8		133	133	133	133	122
15	0.0	0.0		113	113	111	111	111
14	0.2	0.0		313	313	111	111	111
13	0.6	0.0		313	313	111	111	111
12	0.6	0.4		333	222	222	222	222
11	0.8	0.8		333	333	222	222	222
10	0.0	0.8		133	133	133	133	122
9	0.0	0.2		133	133	111	111	111
8	0.0	0.2		133	133	111	111	111
7	0.0	0.2		133	111	111	111	111
6	0.8	0.6		333	111	111	111	111
5	0.8	0.6		333	333	333	333	111
4	0.8	0.6		333	333	333	222	222
3	0.2	0.8		333	333	333	222	222
2	0.2	0.0		313	313	111	111	111
1	0.0	0.0		113	113	111	111	111
			OPTIMUM INITIAL STATE	3	3	1	1	1
			MINIMUM TOTAL EXPECTED COST	0.0	9.3	15.7	21.6	26.7

TABLE V-1

taken because we know that at t=20 there will be no request coming from computer 1. Nevertheless we could, in any case, remain in state 3 without increasing the cost, so both decisions give the same cost.

As we increase the storage cost the number of optimal decisions that are different from "go to state 3" is larger and finally when the storage cost is equal to 1 the optimal decision will always be go to states 1 and 2 only, that is, keeping only one copy in the system at any time.

Looking now at the columns of Table V.1 we see that the optimal cost increases in a nonlinear fashion as the storage cost increases.

For comparison we also consider a static analysis with the corresponding average rates:

$$\theta_1^{av}(t) = \frac{1}{20} \sum_{t=1}^{20} \theta_i(t) = 0.41 \quad \forall t \in [1,20] \qquad (5.1a)$$

$$\theta_2^{av}(t) = \frac{1}{20} \sum_{t=1}^{20} \theta_2(t) = 0.40 \quad \forall t \in [1,20] \qquad (5.2b)$$

We have

storage cost	optimal allocation	total cost per 1,T
0	two copies	0
0.25	two copies	10
0.50	one copy at comp. 1	18
0.75	one copy at comp. 1	23
1.0	one copy at comp. 1	28

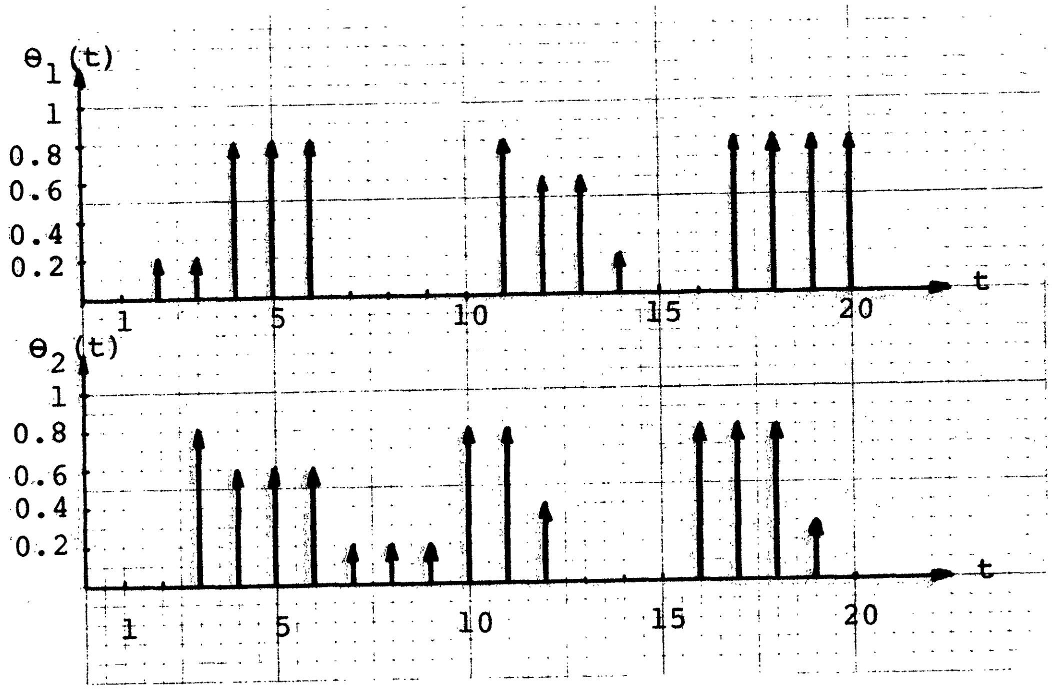


Fig. V.1. Rates. Case 1.

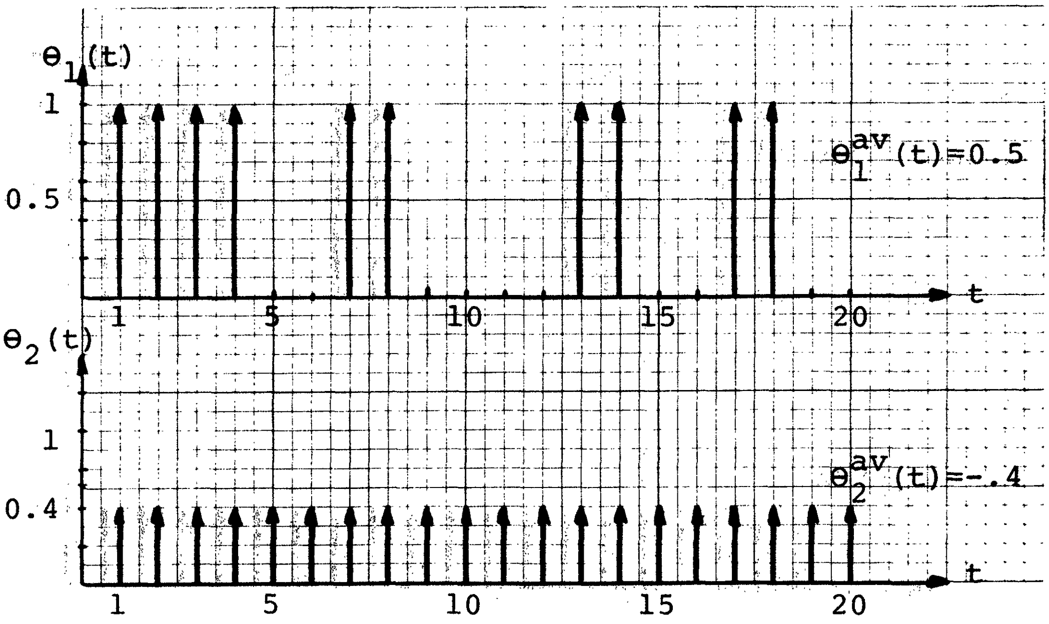


Fig. V.3. Rates. Case 2.

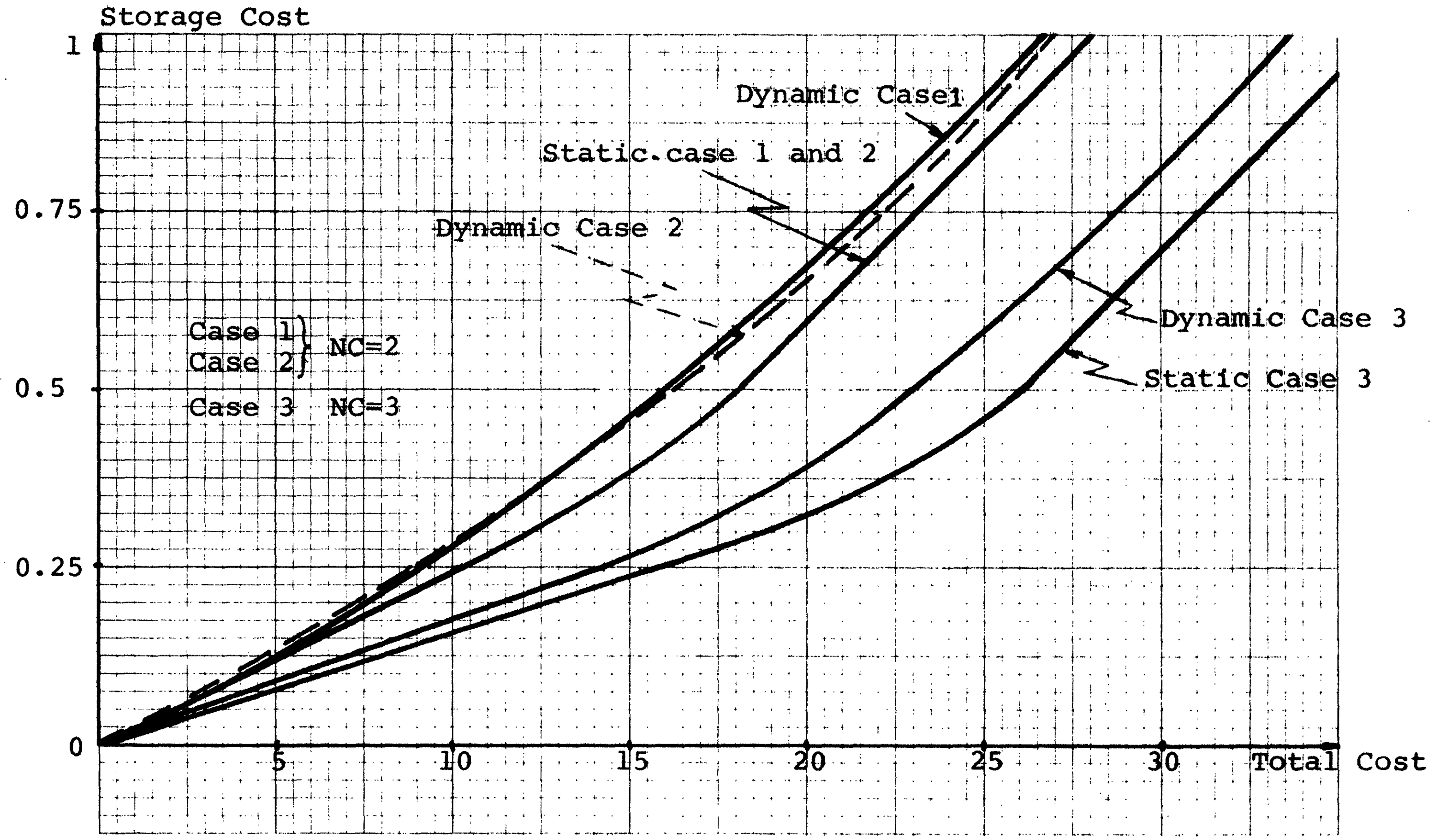


Fig. V.2. Cost Curves for Static and Dynamic Approximations. T=20.

The corresponding total costs are larger here, but not significantly. This is because we are considering a very short period of operation.

The curves describing the evolution of the total cost as the storage cost varies are represented in Fig. V.2 for the state and dynamic cases. We also represent, in the same figure, the two curves corresponding to the rates of Fig. V.3 and the curves corresponding to a network with 3 computers (NC=3) with rates:

$$\begin{aligned} \theta_1(t) &= \theta_1(t)\big|_{\text{case 1}} \\ \theta_2(t) &= \theta_2(t)\big|_{\text{case 1}} \qquad \forall t \in [1,20] \\ \theta_3(t) &= \theta_2(t)\big|_{\text{case 2}} \end{aligned}$$

We can see that in all cases, they have a similar shape. In particular case 1 and case 2 have the same static curve. This is due to the fact that in both cases $\theta_1^{av} > \theta_2^{av}$, θ_2^{av}(case 1) = θ_2^{av} (case 2) and the optimal allocations are the same for every value of the storage cost, as can be easily checked.

V.2 Constant Rates. Updating Traffic. No Failures.

In a practical case it could be very difficult to specify the rates as detailed as a time variant function, even with piecewise constant shape. It seems more reasonable to model the rates as constant functions over a period of time.

An intermediate case will be to obtain the rates as piecewise constant functions with long steps. In this case a quasi-dynamic analysis applying the optimization procedure to every long step separately can be considered. In any case it is important to analyze carefully the behavior of the system with constant rates of demand.

Let us suppose that we have a network with three computers (NC=3), with demand rates

$$\theta_1(t)=0.8 \quad \theta_2(t)=0.6 \quad \theta_3(t)=0.4 \quad \forall t\ 1,8$$

and an operating period equal to eight time units (T=8). As before we consider the transmission cost equal to one for every possible transmission

$$C_{ij}=1 \quad \forall i,j \epsilon \{1,2,3\}$$

The storage costs and updating ratios are the same for all computers

$$\begin{array}{l} C_i = C_s \\ \phi_i = \rho \end{array} \quad \forall i \in \{1,2,3\}$$

and we want to analyze the system for the values

$$C_s = 0,\ 0.25,\ 0.5,\ 0.75,\ 1$$

$$\rho = 0,\ 0.25,\ 0.5,\ 0.75,\ 1$$

The results for $\rho = 0.25$ and $C_s = 0, 0.25, 0.50$ are shown in Tables V-2. We will represent later the evolution of the total cost as the storage cost varies taking ρ as a parameter. Again we will make linear interpolation between exact points.

$\Theta_1=0.8 \quad \Theta_2=0.6 \quad \Theta_3=0.4$

$C_s=0.0 \quad \rho=0.25$

Time	Decision													
	1	2	3	4	5	6	7	1	2	3	4	5	6	7
8								1.40	1.20	0.80	1.00	0.60	0.40	0.00
7	7	7	7	7	7	7	7	2.53	2.24	1.81	2.05	1.62	1.33	0.90
6	7	7	7	7	7	7	7	3.50	3.18	2.73	3.02	2.57	2.25	1.80
5	7	7	7	7	7	7	7	4.42	4.10	3.64	3.95	3.49	3.16	2.70
4	7	7	7	7	7	7	7	5.33	5.00	4.54	4.86	4.39	4.07	3.60
3	7	7	7	7	7	7	7	6.24	5.91	5.44	5.77	5.30	4.97	4.50
2	7	7	7	7	7	7	7	7.14	6.81	6.34	6.67	6.20	5.87	5.40
1	7	7	7	7	7	7	7	8.04	7.71	7.24	7.57	7.10	6.77	6.30

$C_s=0.25 \quad \rho=0.25$

Time	Decision													
	1	2	3	4	5	6	7	1	2	3	4	5	6	7
8								1.65	1.45	1.30	1.25	1.10	0.90	0.75
7	6	6	6	6	6	6	6	3.20	2.96	2.81	2.72	2.57	2.35	2.20
6	6	6	6	6	6	6	6	4.67	4.42	4.27	4.18	4.03	3.80	3.65
5	6	6	6	6	6	6	6	6.12	5.87	5.72	5.63	5.48	5.25	5.10
4	6	6	6	6	6	6	6	7.57	7.32	7.17	7.08	6.93	6.70	6.55
3	6	6	6	6	6	6	6	9.03	8.77	8.62	8.53	8.38	8.15	8.00
2	6	6	6	6	6	6	6	10.48	10.22	10.07	9.98	9.83	9.60	9.45
1	6	6	6	6	6	6	6	11.93	11.67	11.52	11.43	11.28	11.05	10.90

$C_s=0.5 \quad \rho=0.25$

Time	Decision													
	1	2	3	4	5	6	7	1	2	3	4	5	6	7
8								1.90	1.70	1.80	1.50	1.60	1.40	1.50
7	4	4	4	4	4	4	4	3.75	3.50	3.69	3.25	3.35	3.15	3.25
6	4	4	4	4	4	4	4	5.52	5.26	5.47	5.00	5.10	4.90	5.00
5	4	4	4	4	4	4	4	7.27	7.01	7.22	6.75	6.85	6.65	6.75
4	4	4	4	4	4	4	4	9.02	8.76	8.97	8.50	8.60	8.40	8.50
3	4	4	4	4	4	4	4	10.77	10.51	10.72	10.25	10.35	10.15	10.25
2	4	4	4	4	4	4	4	12.52	12.26	12.47	12.00	12.10	11.90	12.00
1	4	4	4	4	4	4	4	14.27	14.01	14.22	13.75	13.85	13.65	13.75

Table V.2.

The first important characteristic that appears looking at these tables is that there are no transients in the decisions. This is not a general statement that can be applied to any network and parameters but will motivate us for a deeper analysis of the facts that are taking place in the optimization algorithm with the assumptions of this section. With these assumptions we observe:

a) The vector $\Lambda(t)$ is time invariant

$\Lambda(t) = \Lambda \; \forall t \in [1,T]$

b) The transition matrix $P(t,u^*)$ corresponding to optimal decision is also time invariant

$P(t,u^*) = P(u^*)$

c) The updating ratio vector (that had to be added to cost-to-go vectors) is also constant. Let us call this vector R

d) The terminal cost vector is equal to the vector Λ

$V(T) = \Lambda(T) = \Lambda$

Therefore we can write the recursive equation for the first iteration

$$V(T-1) = \Lambda + P(u_1^*)\;(\Lambda+R) \qquad (5.2)$$

We want to show first that with these conditions, the system exhibits a trapping state.

Let us call $\Lambda+R = \Omega^1$ where $\Omega^1 = [\omega_1^1, \omega_2^1, \ldots \omega_M^1]^T$

and let us suppose that ω_i is the smallest component of vector Ω^1

$$\omega_i^1 < \omega_j^1 \qquad \forall j \neq i \quad j \in \{1,2,\ldots M\} \qquad (5.3)$$

For the system can remain in the same state, if this is the decision, with probability one, the ith row (corresponding to state i) of $P(u_1^*)$ will be then

$$\text{ith row} \longrightarrow (0\ 0\ 0\ -\ -\ 0\ \overset{i}{1}\ 0\ -\ -\ 0) \qquad (5.4)$$

furthermore if we represent by K the set of states having the same 1 components as state i and at least one 1 component more (we will call this set the optimum set) we have

$$\text{kth row of } P(u_1^*) = (0\ 0\ -\ -\ 0\ \overset{i}{1}\ 0\ -\ -\ 0)\ \forall k \in K \qquad (5.5)$$

if we call now

$$P(u_1^*)\Omega^1 = \Phi^1 \text{ where } \Phi^1 = (\phi_1^1, \phi_2^1, ..\phi_M^1)^T \qquad (5.6)$$

we have

$$\begin{aligned} &\phi_l^1 = \omega_i^1 \quad \forall\ l \in K \text{ or } l=i \\ &\phi_m^1 > \phi_l^1 \quad \forall m \in \{1,2,..M\}, m \neq l \end{aligned} \qquad (5.7)$$

Writing now the second backward step in the recursive equation we have

$$\begin{aligned} V(T-2) &= \Lambda + P(u_2^*)\ [V(T-1) + \bar{R}] = \\ &= \Lambda + P(u_2^*)\ [\Lambda + \Phi^1 + R] = \Lambda + P(u_2^*)\Omega^2 \end{aligned} \qquad (5.8)$$

where

$$\Omega^2 = \Lambda + \Phi^1 + R = \Omega^1 + \Phi^1 \qquad (5.9)$$

from (5.3) and (5.7) we have again

$$\omega_i^2 < \omega_j^2\ \forall_j \neq i \quad j \in \{1,2,..M\} \qquad (5.10)$$

and hence

$$\text{kth row of } P(u_2^*) = (0\ 0\ -\ -\ 0\ \overset{i}{1}\ 0\ -\ -\ 0)\ \forall k \in K \text{ or } k=i \qquad (5.11)$$

i.e. the optimal decision from state i is again "remain in state i"; and this decision can be implemented with probability one. From this result we obtain, as before

$$\phi_l^2 = \omega_i^2 \qquad \forall l \in k \qquad \text{or} \qquad l = i \qquad (5.12)$$

$$\varepsilon_m^2 > \phi_l^2 \qquad \forall m \in \{1,2,..M\} \qquad m \neq l \qquad (5.13)$$

where

$$P(u_2^*)\ \Omega^2 = \Phi^2 = (\phi_1^2, \phi_2^2, ..\phi_M^2)^T \qquad (5.14)$$

This characteristic or behavior will be repeated through the remaining steps. Therefore state i will be an absorbing or trapping state, according to the most frequent nomination for these kinds of states in the literature (see refs. [18] and [19]). This means that once the system visits i it remains there forever. Furthermore if the process is long enough we may expect that the optimal initial (at t=1) allocation will correspond to state i or any state $k \in K$ and thus the system will fall off in state i since the very beginning.

The former property suggests a very efficient and quasi-optimal procedure to analyze a system with the above characteristics. We can describe this procedure as follows:

A quasi-optimal steady-state ($\forall t\ [1,T]$) decision for a system with constant parameters and no failures will be to allocate the file according to the description of state i, with i obtained from the condition

$$\omega_i < \omega_J \qquad \forall j \neq i; j \in \{1,2,..M\} \qquad (5.15)$$

where

$$[\omega_1, \omega_2, ..\omega_M]^T = \Omega = \Lambda + R$$

This quasi-optimal decision role can be considerably far from the true optimum if the terminal costs were very different in value from the vector Λ, and if the operating period were too short as to be able to disguise the influence of the terminal costs. We can see that this decision is in fact the decision we had arrived to if the analysis had been made under the so called static approximation.

Although the former decision could be enough for our purposes, since the concept of trapping state is conclusive in the performance of this kind of process, it does not clarify completely the lack of transients in the iterations. An intuitive and heuristic idea in this direction may come thinking that the optimization process will be very much biased by the states belonging to the set K defined above. This bias will be in the sense that any state not in the set K will try to move toward some of the states belonging to K in order to reduce the cost. (In some sense this set K could be interpretated as a "stopping set" using De Leve terminology [21]). This behavior is represented graphically in Fig. V.4.

In particular, for the case $\rho=0.25$, $C_s=0.25$ of the example we are considering, Table V.2, the diagram of optimal transitions is represented in Fig. V.5.

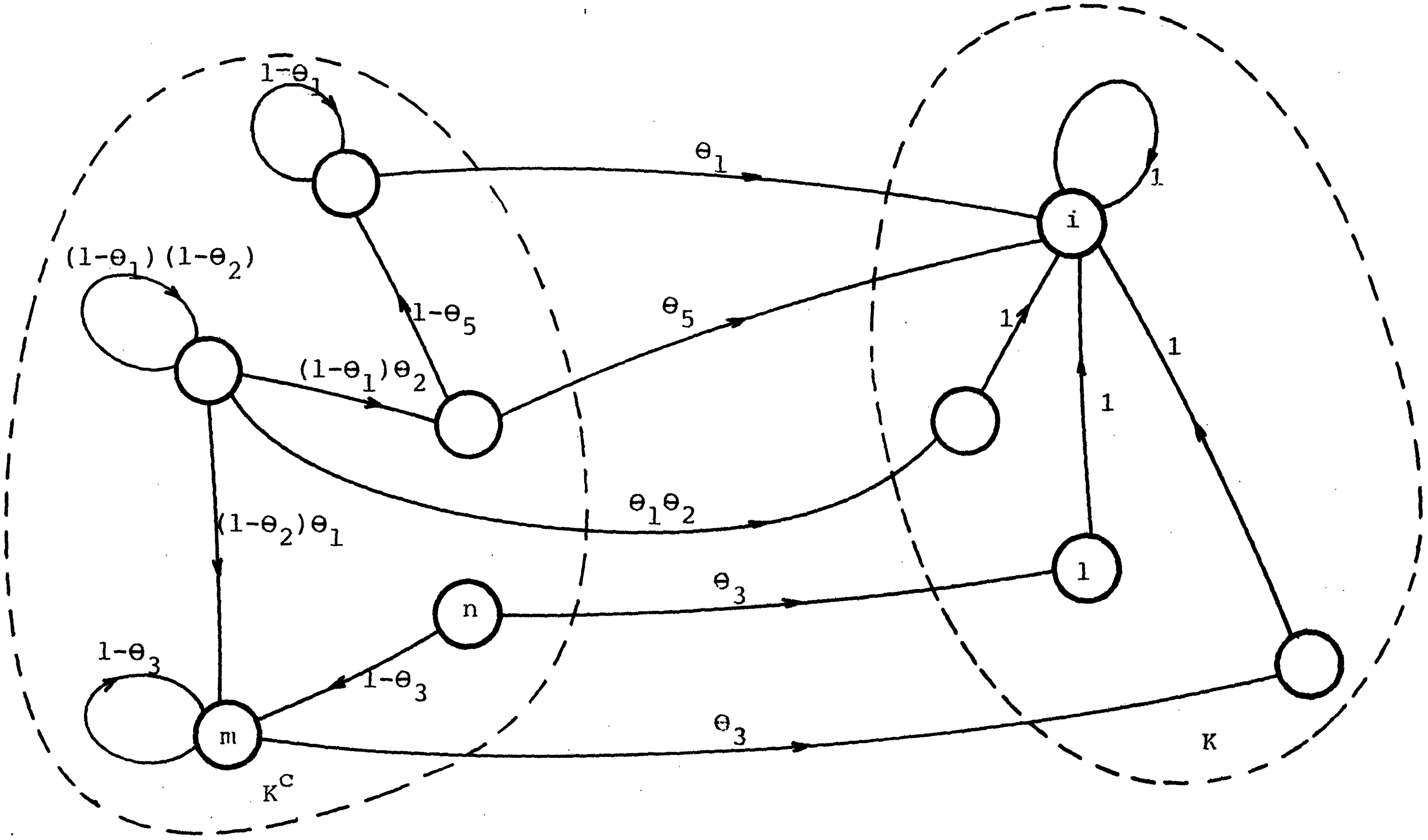


Fig. V.4. Qualitative sketch of a typical behavior for a system with constant parameters. K is the optimal set and K^c its complementary (only some states are represented).

$$K=\begin{cases}6 \text{ —— } 1\ 1\ 0\\ 7 \text{ —— } 1\ 1\ 1\end{cases}\qquad K^c=\begin{cases}1 - 0\ 0\ 1\\ 2 - 0\ 1\ 0\\ 3 - 0\ 1\ 1\\ 4 - 1\ 0\ 0\\ 5 - 1\ 0\ 1\end{cases}\tag{5.16}$$

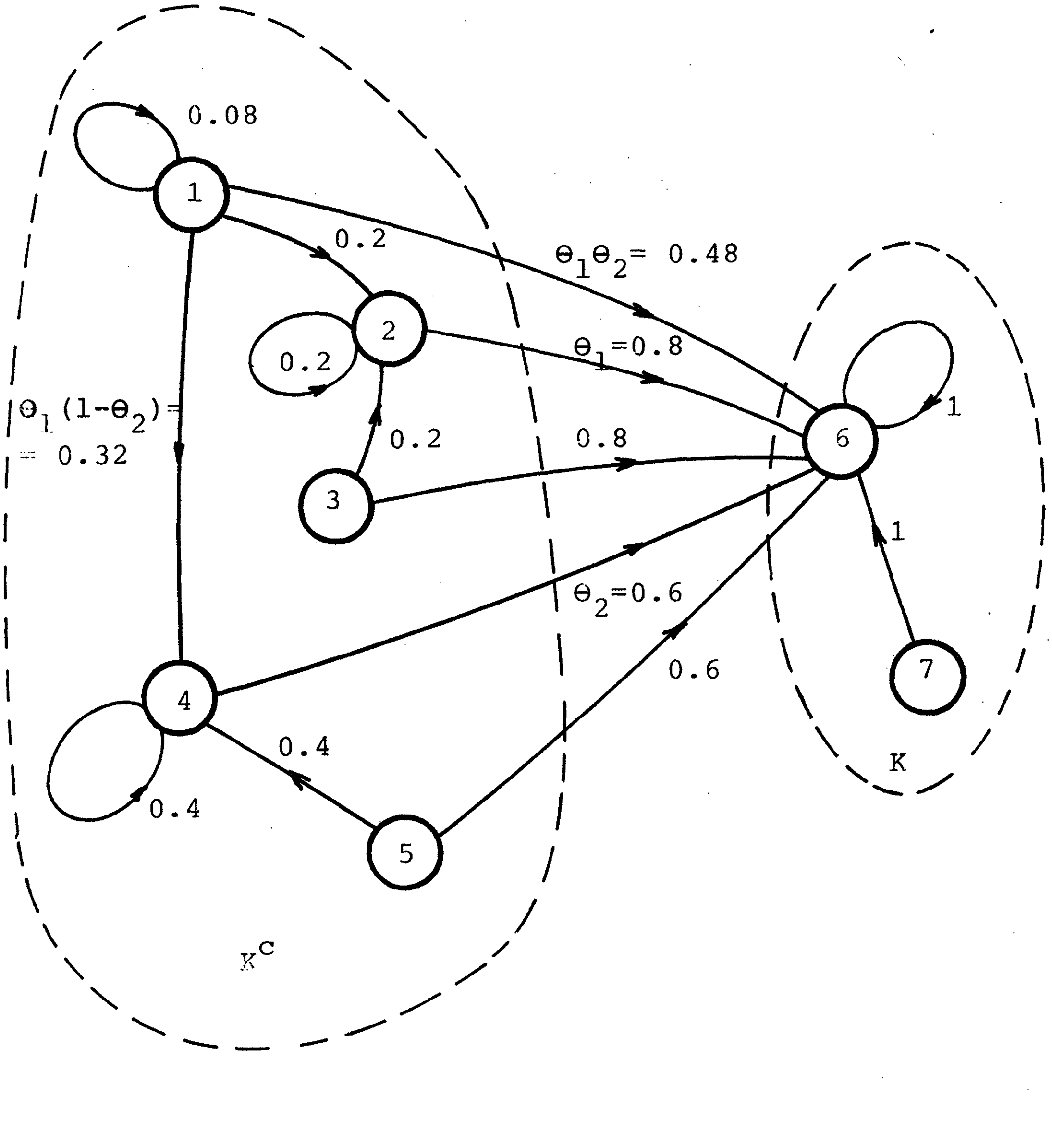


Fig. V.5 Optimal transitions for the example of this section with P=0.25, C_s=0.25.

Let us go back to Table V.2 and look at the minimum cost and optimal initial states . We have worked out the minimum total cost for every instant of time, and the optimal initial states assuming that the process might begin at some time, not necessarily at time 1. We can see several interesting results looking at those marked numbers of the table, namely: For a given parameter values:

a) The minimum total cost up to any instant of time always corresponds to the same initial state.

b) Consequently the optimum initial state is always the same for every time.

c) The increasing of the minimum total cost is linear with time.

d) The optimum initial state does not necessarily coincide with the omnipresent optimum decision.

Do these conclusions generalize to any system with constant parameters? Let us investigate this question.

From previous discussions it is clear that the optimum initial state will belong to the optimum set of states, the set K. Remember that if the terminal costs were different from the p.u.t. costs we had to wait until the steady state were reached in order to make this statement; otherwise the optimum initial state does not need to be in the optimum set. For the states belonging to the set K we can write the recursive equation as follows:

$$V\ (t) = \lambda_k + V_i(t+1) + r_i \quad \begin{matrix} \forall k \in K \\ i \text{ (trapping state)} \in K \end{matrix} \qquad (5.17)$$

and for the next step

$$V_k(t-1) = \lambda_k + V_i(t) + r_i \tag{5.18}$$

therefore

$$V_k(t-1) - V_k(t) = V_i(t) - V_i(t+1) = \lambda_i + r_i = \omega_i \tag{5.19}$$

but

$$V_k(T) = \lambda_k \tag{5.20}$$

so we can write

$$V_k(t) = \lambda_k + (T-t)\ \omega_i \qquad \forall k \in K \tag{5.21}$$

This important result will answer many of our questions. In fact

$$\lambda_j < \lambda_k \quad \begin{matrix} \forall\ k,j \in K \\ k \neq j \end{matrix} \quad \Longrightarrow \quad V_j(t) < V_k(t) \quad \begin{matrix} \forall k,j \in K \\ k \neq j \\ \forall t \in [1,T] \end{matrix}$$

Hence the minimum total cost will always correspond to the same state no matter how long the operating period is. Obviously that state will be the optimum initial state for any instant of time.

Furthermore, the minimum total cost increases linearly with time as it was expected.

So far we have proved that facts a), b) and c) are valid for any system with constant parameters. Concerning fact d) it is clear that the state with minimum per unit time cost among the optimum set need not be the trapping state, but if the optimum set only has one element (as in the case $C_s=0$, $\rho=0.25$) then it is clear that this element has to be the trapping state.

We can see that the problem with constant parameters and terminal costs equal Λ falls in the section of problems where the optimum decision is the decision that minimizes the immediate cost.

Before leaving Table V.2 it can be useful to make a few more comments. For instance we will not always find a unique optimal decision, no matter which state or time we are, as the one shown in that table; as an example the case $C_s=1$, $\rho=0.5$, below shows a transition from 3 to 2 as intermediate step to arrive to state 4.

Table V.3

$\theta_1=0.80$ $\theta_2=0.60$ $\theta_3=0.40$ $\rho=0.5$ $C_s=1$

Time	Decision													
	1	2	3	4	5	6	7	1	2	3	4	5	6	7
8								2.40	2.20	2.80	2.00	2.60	2.40	3.00
7	4	4	2	4	4	4	4	5.02	4.76	5.60	4.50	5.10	4.90	5.50
6	4	4	2	4	4	4	4	7.54	7.27	8.16	7.00	7.60	7.40	8.00
5	4	4	2	4	4	4	4	10.05	9.77	10.67	9.50	10.10	9.90	10.50
4	4	4	2	4	4	4	4	12.55	12.27	13.17	12.00	12.60	12.40	13.00
3	4	4	2	4	4	4	4	15.05	14.77	15.67	14.50	15.10	14.90	15.50
2	4	4	2	4	4	4	4	17.55	17.27	18.17	17.00	17.60	17.40	18.00
1	4	4	2	4	4	4	4	20.05	19.77	20.67	19.50	20.10	19.90	20.50

This is due to the fact that a decision of moving directly from 3(0 1 1) to 4(1 0 0) has some intrinsic risk of remaining in 3 if no request is made from computer 1. It turns out that in this case a large price has to be paid if the system remains at 3. On the other hand a transition to state 2 can be done with probability one and carries with it a smaller cost. It can be expected that in a more general case with more computers and consequently a larger variety of states this event will appear more frequently.

An important observation is to note that ω_i can be identified with the steady state gain of the process as defined by Howard [20]. Equation (5.21) for optimum cost-to-go will justify our assertion if we can show that the gain is unique even under optimal policies that make the Markov process non ergodic.

The only type of policies that can make the system non ergodic (in the sense defined by Howard) will be those containing at least two persistent decision of the form "remain in i_j" where now each i_j will be a trapping state. We will show that if l and m are two of these trapping states then

$$\omega_l = \omega_m$$

To see that let us call K(l) and K(m) to the corresponding optimal sets. Notice that

$$K(l) \cap K(m) \neq \Phi \qquad (\Phi \equiv \text{empty set})$$

because K(l) K(m) will contain at least the element M=(1 1...1). Therefore if

$$r \in K(l) \cap K(m)$$

when

$$V_r(t) = \lambda_r(t) + (T-t)\ \omega_l =$$
$$= \lambda_r(t) + (T-t)\ \omega_m$$

and hence

$$\omega_1 = \omega_m$$

We conclude that the gain is always unique, even in the case that the policy would make the system non cryodic.

The discontinuous staircase line in Table V.2 represents the beginning of a constant increase in costs for every state of the process.

Let us analyze now the evolution of the total cost versus storage cost taking the updating ratio as a parameter.

The curves are shown in Fig. V.6.

First of all we see that the shape of the curves is similar to the one obtained in section V.1 for the time varying case without updating traffic. The curves present a larger curvature for small values of C_s and ρ and then the behavior is almost linear. In Fig. V.7 we represent the total minimum cost versus ρ taking now C_s as a parameter. We can see that both sets of curves are quite close in shape but the curves of total cost versus ρ have a deeper slope and after the curvature section they are completely linear. We could obtain a better understanding of this curve if we divide the whole quadrant in three sections corresponding each section to trapping states with the same number of copies. In this case we will have a section for three copies, other section for trapping state with two copies and the third one for only one copy.

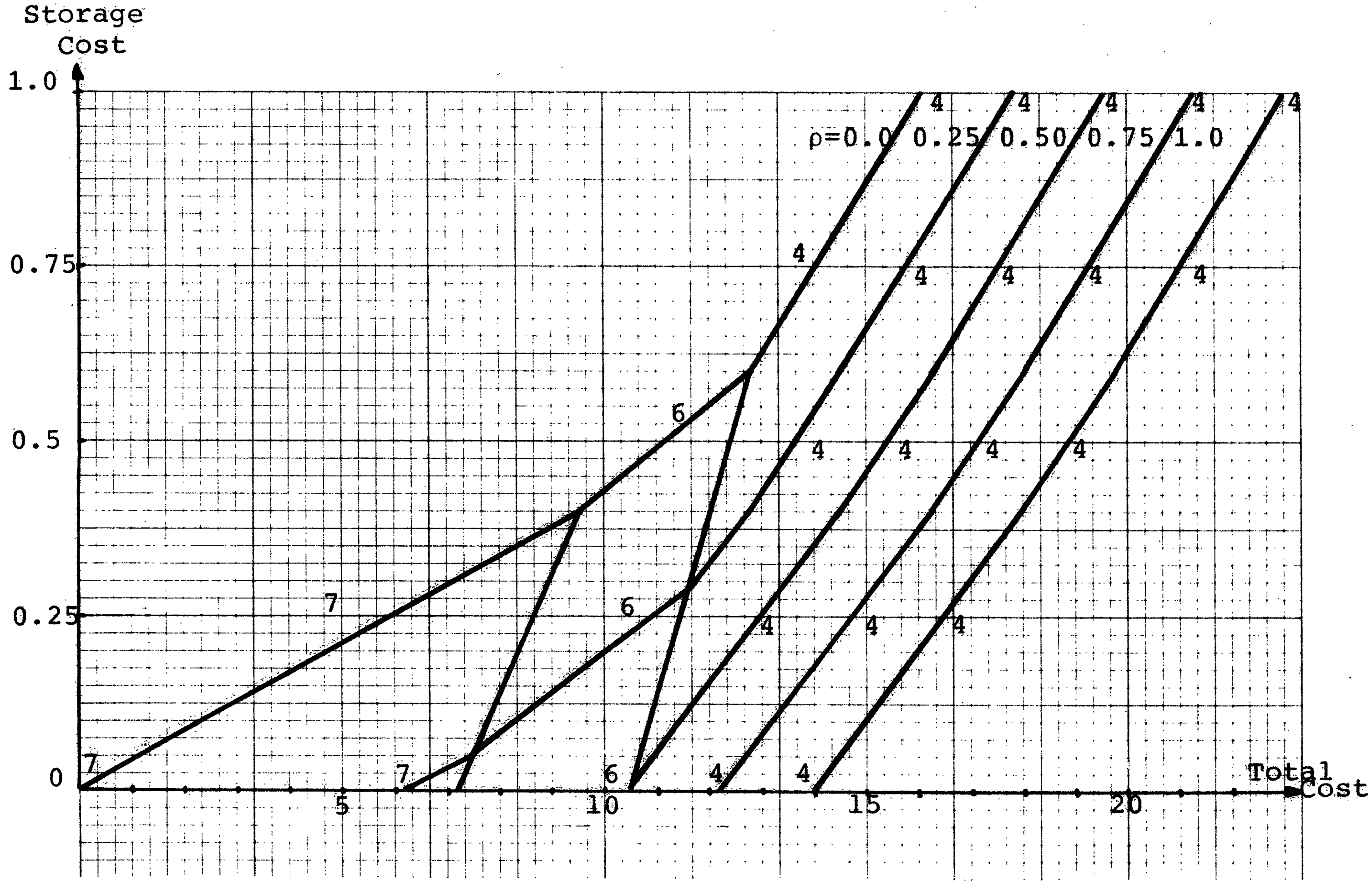


Fig. V.6. Total cost versus Storage Cost with ρ as a parameter. (Linear interpolation between exact points.)

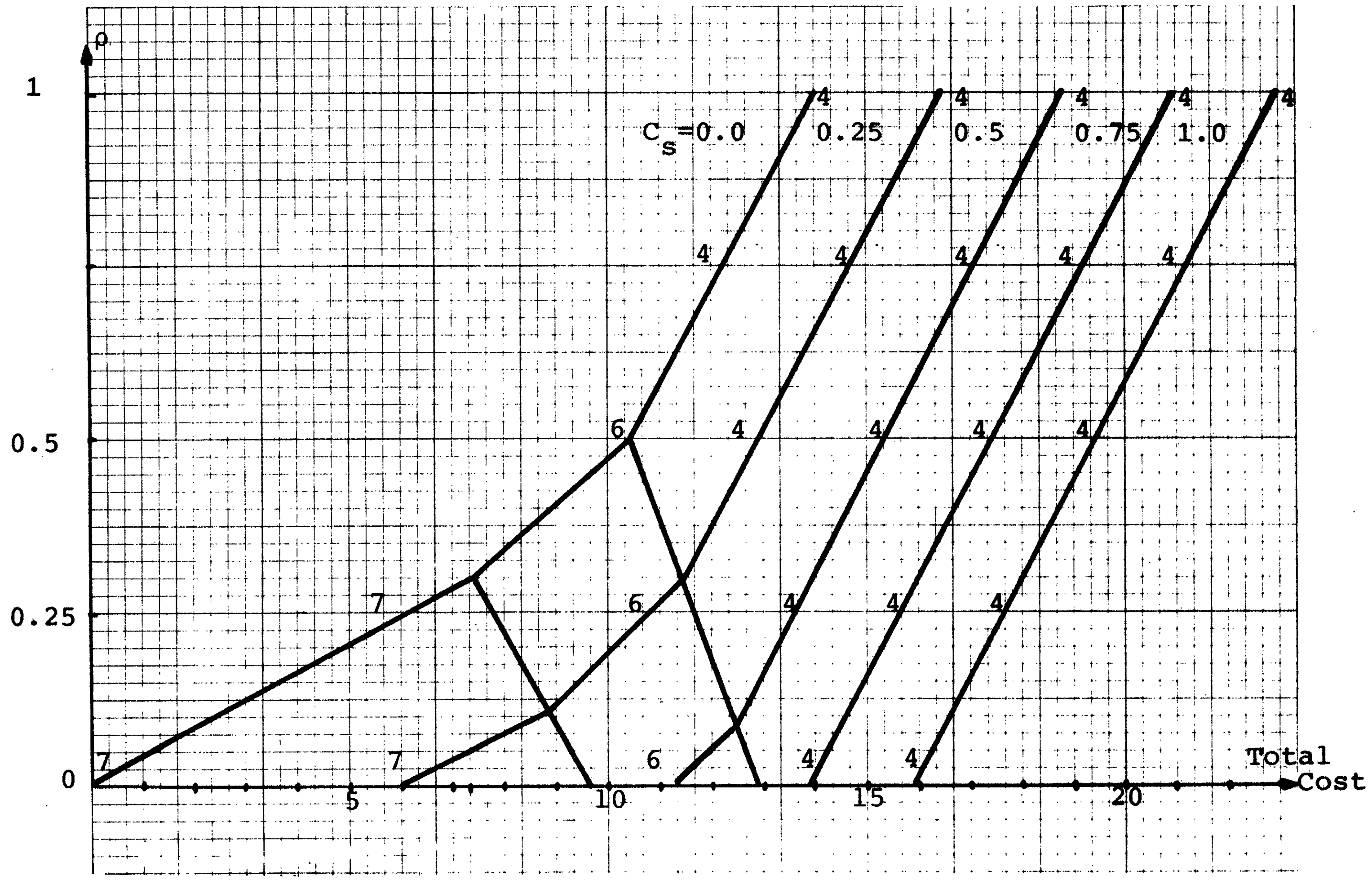


Fig. V.7. Total Cost vs. ρ with C_s as a parameter.

At each of the points of the curves, corresponding to the cases studied in the present example, we have encircled the trapping state corresponding to the system with the particular parameter assigned to the point. We can see now that linear (or almost linear) part of the curves corresponds to points with the same trapping state while the rounded parts are due to changes in this state.

We could now ask the following question regarding the trapping states. Why the trapping state, having two copies, is always state 6(110), and why the trapping state having one copy is always 4(100)? The reason is very simple. Remember that our rates were such that

$$\theta_3 = 0.4 < \theta_2 = 0.6 < \theta_1 = 0.8 \tag{5.26}$$

therefore, other parameters being equal, the system will try to keep copies in the computer with higher request rate and there is no reason why it should behave in a different way. This fact is easily generalizable and we can say that if we enumerate the computers according to the sequence of decreasing values of the rates

$$\theta_1 \geq \theta_2 \geq \theta_3 \geq \theta_4 \geq \text{----} \geq \theta_{NC} > 0 \tag{5.27}$$

the trapping state will always have its "1" components concentrated in the left of its vector representation.

Let us now analyze for what values of C_s and ρ will the trapping state change. Concentrating first on the case $\rho=0$ we can see that the change from trapping state=7 to trapping state=6 will be at some point in between C_s=0.25 and C_s=0.5. We claim that the frontier will be marked by the point at which the system, being in state 6, is indifferent to a transition to state 7 to as opposed to remaining in state 6. A small perturbation from this equilibrium point will require the system to go to 7 (if the perturbation is to the left) or to remain in 6 (if the perturbation is to the right).

The reason is again very simple. State 6 cannot be a trapping state if being in this state, the system decides to move to some other state. On the othet hand, once the system decides to remain in 6, this state, considering all the decisions above, becomes a trapping state.

Let us find this equilibrium point. Consider that the system is at state 6 ≡ (1 1 0) at time t-1, then we have:

Cost due to a transition from 6 to 6

$$\phi(6.6) = V_6(t) + r_6 \tag{5.28}$$

Cost due to a transition from 6 to 7 ≡ (1 1 1)

$$\phi(6.7) = \Theta_3\left[V_7(t)+r_7\right] + (1-\Theta_3)\left[V_6(t)+r_6\right] \tag{5.29}$$

Equilibrium point (assuming $\Theta_3 \neq 0$)

$$\phi(6.6) = \phi(6.7) \iff V_7(t) + r_7 = V_6(t) + r_6 \tag{5.30}$$

but we know from (5.21) that

$$V_7(t) = \lambda_7 + (T-t)\ \omega_i \tag{5.31}$$

$$V_6(t) = \lambda_6 + (T-t)\ \omega_i$$

therefore the equilibrium point will be at

$$\lambda_7 + r_7 = \lambda_6 + r_6 \tag{5.32}$$

(notice that in fact $\lambda_6 + r_6 = \lambda_7 + r_7 = \omega_i$)

$$\lambda_7 = 3C_s \qquad r_7=2\rho(\Theta_1+\Theta_2+\Theta_3)$$
$$\lambda_6 = 2C_s+\Theta_3 \qquad r_6=\rho(\Theta_1+\Theta_2+2\Theta_3) \qquad (5.33)$$

and this substitution yields

$$\lambda_7 + r_7 = \lambda_6 + r_6 \Longleftrightarrow \quad C_s + \rho(\Theta_1 + \Theta_2) = \Theta_3 \qquad (5.34)$$

So we have obtained a linear relation between ρ and C_s that describes the equilibrium point. For $\rho=0$ the equilibrium is at

$$C_s = \Theta_3 = 0.4 \qquad (5.35)$$

For $\rho = 0.25$ the equilibrium is at

$$C_s = \Theta_3 - 0.25\ (\Theta_1+\Theta_2) = 0.4 - 0.35 = 0.05 \qquad (5.36)$$

The intersection of the equilibrium line with the horizontal axis will correspond to the cost due to the equilibrium point with $C_s = 0$

$$C_s = 0 \implies \rho = \Theta_3/\Theta_1+\Theta_2 = 0.29 \qquad (5.37)$$

$$V_7(1)\Bigg|_{\substack{C_s=0\\ \rho=0.29}} = (8-1)\omega_i = 7(\lambda_7+r_7) = 7.20 \qquad (5.38)$$

We can see that for the given rates the equilibrium above is possible only if $\rho \leq 0.29$

With the same reasons as before we claim now that the equilibrium between trapping state 6(110) and trapping state 4(100) will be defined by the points at which the system, being in state 4, is indifferent to a movement toward state 6 or staying in state 4.

The new equilibrium line will be defined by

$$\phi(4,4) = \phi(4,6) \tag{5.39}$$

where

$$\phi(4,4) = V_4(t) + r_4 \tag{5.40}$$

$$\phi(4,6) = \theta_2\left[V_6(t) + V_6\right] + (1-\theta_2)\left[V_4(t) + rV_4\right]$$

then

$$\phi(4,4) = \phi(4,6) \Longleftrightarrow V_4(t) + r_4 = V_6(t) + r_6 \tag{5.41}$$

but

$$V_4(t) = \lambda_4 + (T-t)\,\omega_i \tag{5.42}$$

$$V_6(t) = \lambda_6 + (T-t)\,\omega_i$$

with

$$\lambda_4 = C_s + \theta_2 + \theta_3 \qquad r_4 = \phi(\theta_2 + \theta_3) \tag{5.43}$$

$$\lambda_6 = 2C_s + \theta_3 \qquad r_6 = \phi(\theta_1+\theta_2+2\theta_3)$$

hence the equilibrium line is defined by

$$\lambda_4 + r_4 = \lambda_6 + r_6 \Longleftrightarrow C_s + \rho(\theta_1+\theta_3) = \theta_2 \tag{5.44}$$

therefore

$$\rho = 0 \Longrightarrow C_s=\theta_2=0.6$$

$$\rho = 0.25 \Longrightarrow C_s=0.3 \tag{5.45}$$

$$\rho = 0.5 \Longrightarrow C_s = 0$$

Now after all these properties have been described we are ready to provide more information about linearity and curvature of the graphs of Fig. V.6 and V.7. But before doing that let us generalize the results obtained above.

We claim that in a network with constant parameters and NC computers numbered in such a way that

$$\theta_1 \geq \theta_2 \geq \theta_3 \geq \cdots \geq \theta_{NC-1} \geq \theta_{NC} \qquad (5.46)$$

the equilibrium line separating the trapping state with NC copies from the trapping state with NC-1 copies will be defined by the relation

$$\phi(M-1,\ M-1) = \ (M-1,M) \qquad (5.47)$$

where

$$M=2^{NC}-1$$

Similarly the equilibrium line separating the trapping state with NC-1 copies from the trapping state with NC-2 copies will be defined by the relation

$$\phi(\mu,\mu) = \phi(\mu,\ M-1) \qquad (5.48)$$

where μ is described by the vector (1,1,1,..1,0,0)

$$\text{i.e. } \mu = \sum_{n=2}^{NC} 2^n = M-1-2 = M-3 \qquad (5.49)$$

In general the equilibrium line separating trapping states with n and n-1 number of copies will be described by the relation

$$\phi(\nu,\nu) = \phi(\nu,\nu + 2)$$

where

$$\nu \equiv (\underbrace{1,\ 1,\ ..\ 1}_{n-1},\ \underbrace{0,\ ..\ 0}_{NC-n+1}) \quad ,\ \text{i.e.} \nu = \sum_{m=NC-n}^{NC} 2^m \qquad (5.50)$$

We can arrive to an explicit relation of this equilibrium line as a function of the parameter of the system if we replace $\phi(.,.)$ by its expression in function of those parameters. In fact if we are in state ν at time t-1 we have

$$\phi\ (\nu,\nu) = V_\nu(t) + r_\nu$$
$$\phi\ (\nu,\nu+2) = \theta_n\left[V_{\nu+2}(t)+r_{\nu+2}\right] + (1-\theta_n)\left[V_\nu(t)+r_\nu\right] \qquad (5.51)$$

therefore we will have equilibrium if and only if

$$V_\nu(t) + r_\nu = V_{\nu+2}(t) + r_{\nu+2} \qquad (5.52)$$

but remembering that

$$V_k(t) = \lambda_k + (T-t)\ \omega_i \quad \forall k \in K \underset{i\in K}{} \quad k \neq i \qquad (5.53)$$

we will have equilibrium if and only if

$$\lambda_\nu + r_\nu = \lambda_{\nu+2} + r_{\nu+2}$$

but from (3.6) and (4.1) we know that

$$\lambda_\nu = (n-1)\ C_s + \sum_{m=n}^{NC} \theta_m \qquad (5.55)$$

$$r_\nu = \rho\left[(n-2)\sum_{m=1}^{n-1} \theta_m + (n-1)\sum_{m=n}^{NC} \theta_m\right] =$$
$$= \rho\left[(n-2)\sum_{m=1}^{NC} \theta_m + \sum_{m=n}^{NC} \theta_m\right] \qquad (5.56)$$

$$\lambda_{\nu+2} = nC_s + \sum_{m=n+1}^{NC} \theta_m \qquad (5.57)$$

$$r_{\nu+2} = \rho\left[(n-1)\sum_{m=n}^{NC} \theta_m + \sum_{m=n+1}^{NC} \theta_m\right] \qquad (5.58)$$

therefore we will have equilibrium if and only if

$$(n-1)\ C_s + \sum_{m=n}^{NC} \theta_m + \rho\left[(n-2)\sum_{m=1}^{NC} \theta_m + \sum_{m=n}^{NC} \theta_m\right] =$$

$$= nC_s + \sum_{m=n+1}^{NC} \theta_m + \rho\left[(n-1)\sum_{m=1}^{NC} \theta_m + \sum_{m=n+1}^{NC} \theta_m\right] \qquad (5.59)$$

that is iff

$$\theta_n = C_s + P\sum_{\substack{m=1\\ m\neq n}}^{NC} \theta_m \qquad (5.60)$$

Let us now calculate the total minimum expected cost. Clearly this cost will be given by

$$V_j(1) = \lambda_j + (T-1)\ \omega_i \qquad j,\ i \in k \qquad (5.61)$$

where

$$\lambda_j < \lambda_k \quad \forall\ k \neq j \quad k \in K$$

$i = \nu+2$ or $i = \nu$ in the equilibrium line because

$$\omega_i \equiv \lambda_\nu + r_\nu = \lambda_{\nu+2} + r_{\nu+2} \qquad (5.62)$$

and $i = \nu+2$ to the left of this line whereas $i=\nu$ to the right. The minimum cost in the left side of the equilibrium line (without trespassing the area where $\nu+2$ is the trapping state) will be then

$$V_j^1(1) = \lambda_j + \lambda_{\nu+2} + r_{\nu+2} \qquad j \in K^1 \qquad (5.63)$$

whereas in the right side and before crossing another equilibrium line the cost will be

$$V_j^r(1) = \lambda_j + \lambda_\nu + r_\nu \qquad\qquad j\ K^r \qquad\qquad (5.64)$$

It is important to notice that j does not have to be the same in both expressions because when we move the point up and down on the curves, we are changing C_s or ρ and j will be a function of C_s. Furthermore the set K (remember j K) will be increased by one element, the new trapping state, every time we cross an equilibrium line.

Let us express these two costs in terms of the parameters of the system.

$$V^1(1) = \min_{n \le \tau \le NC} \left\{ \tau C_s + \sum_{m=\tau+1}^{NC} \theta_m \right\} + nC_s + A + \rho B \qquad (5.65)$$

$$V^r(1) = \min_{(n-1) \le \tau \le NC} \left\{ \tau C_s + \sum_{m=\tau+1}^{NC} \theta_m \right\} + (n-1)C_s + F + \rho G \qquad (5.66)$$

where A and B are constant (in the region where $\nu+2$ is the trapping state) with values

$$A = \sum_{m=n+1}^{NC} \theta_m \qquad (5.67)$$

$$B = (n-1) \sum_{m=1}^{NC} \theta_m + \sum_{m=n+1}^{NC} \theta_m \qquad (5.68)$$

and F and G are also constant (in the region where ν is the trapping state) with values

$$F = \sum_{m=n}^{NC} \theta_m = A + \theta_m \qquad (5.69)$$

$$G = (n-2)\sum_{m=1}^{NC} \theta_m + \sum_{m=n}^{NC} \theta_m = B - \sum_{m=1}^{NC} \theta_m + \theta_n \qquad (5.70)$$

Now we have all the necessary elements to study the exact shape of the curves of total cost versus storage cost taking the updating rate as a parameter or otherwise total cost versus updating ratio taking the storage cost as a parameter.

For ρ=const.V*(1) is a picewise continuous linear function of C_s that will change its slope every time we find a new minimum in the term in braces in (5.65) this will happen every time C_s θ_m for $n+1 \le m \le NC$)

For C_s = const. (5.71)

$$\min_{a \le \tau \le NC} \tau C_s + \sum_{m=\tau+1}^{NC} \theta_m = \text{constant for a fixed a}$$

i.e., is constant in the zone between equilibrium lines and therefore V*(1) is a picewise continuous linear function of ρ that will only change its slope every time it crosses an equilibrium line.

With these properties we can now draw the exact cost curves for the example of this section, without implementing the dynamic program algorithm. The curves are in Fig. V.8 a) and b). This figure closes the analysis of networks with constant parameters and updating traffic and in the next section the analysis of the computer networks with constant parameters, updating traffic and some nonzero probability of node failures will be considered.

V.3. Nonzero Failure Probability

The dynamic analysis under consideration takes its complete meaning when we include in the system the probability of node failure.

The Markov process describing the evolution of the system without failures under optimal decision rules has a trapping state, a fact which gives rise to a number of specific properties as described in section V.2. On the other hand, if failures occur, the trapping state disappears, as seen from the discussion in Chapter IV. In particular, if failures can happen in all computers with nonzero probability, then the steady state probability that the Markov process will be in a given state is strictly less than 1 for any decision strategy.

Let us analyze a simple example. Consider a network with two computers, NC=2, and the following parameters:

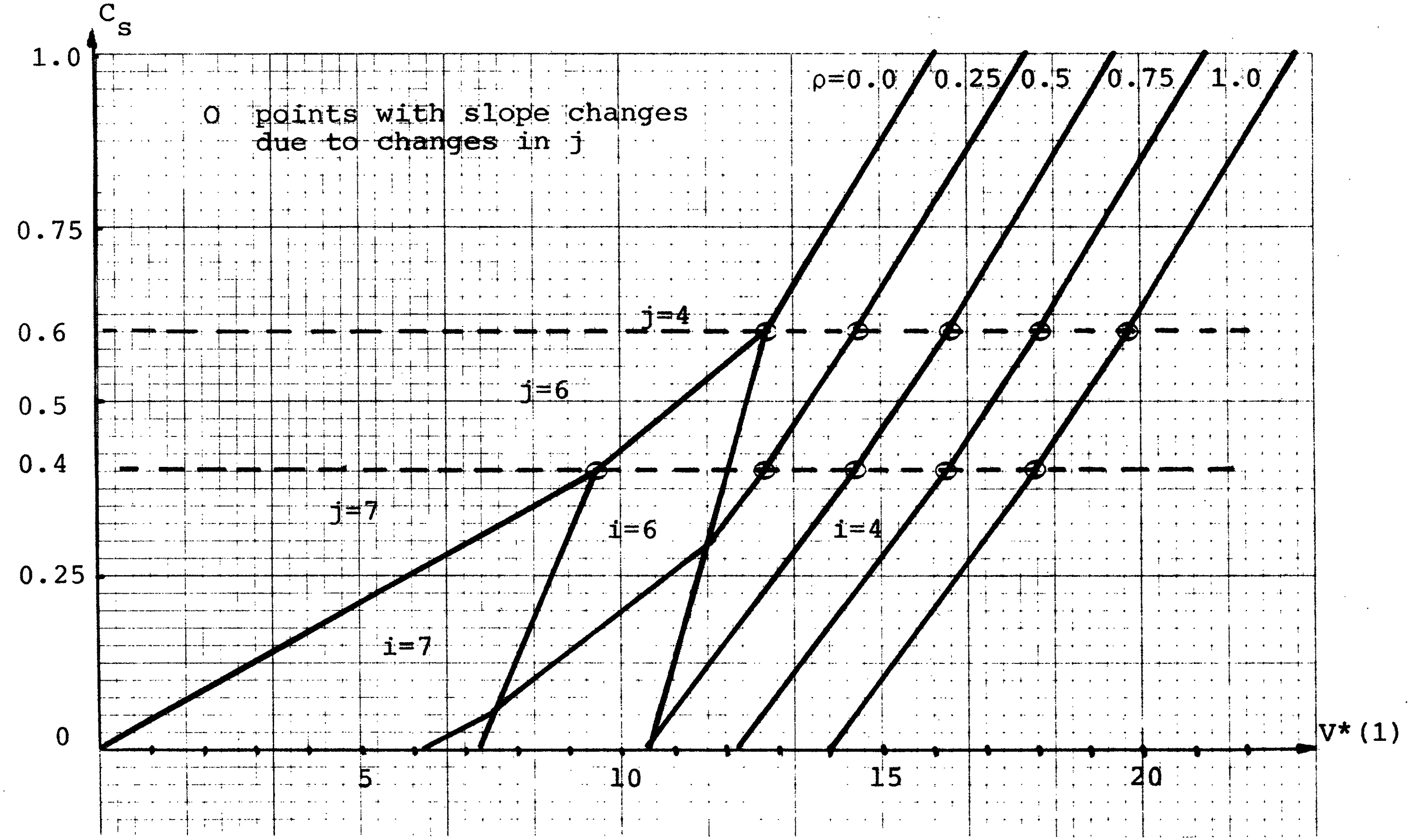


Fig. V.8a. Curves of total minimum cost vs. storage cost with ρ as a parameter. i represents trapping state and j optimum initial state.

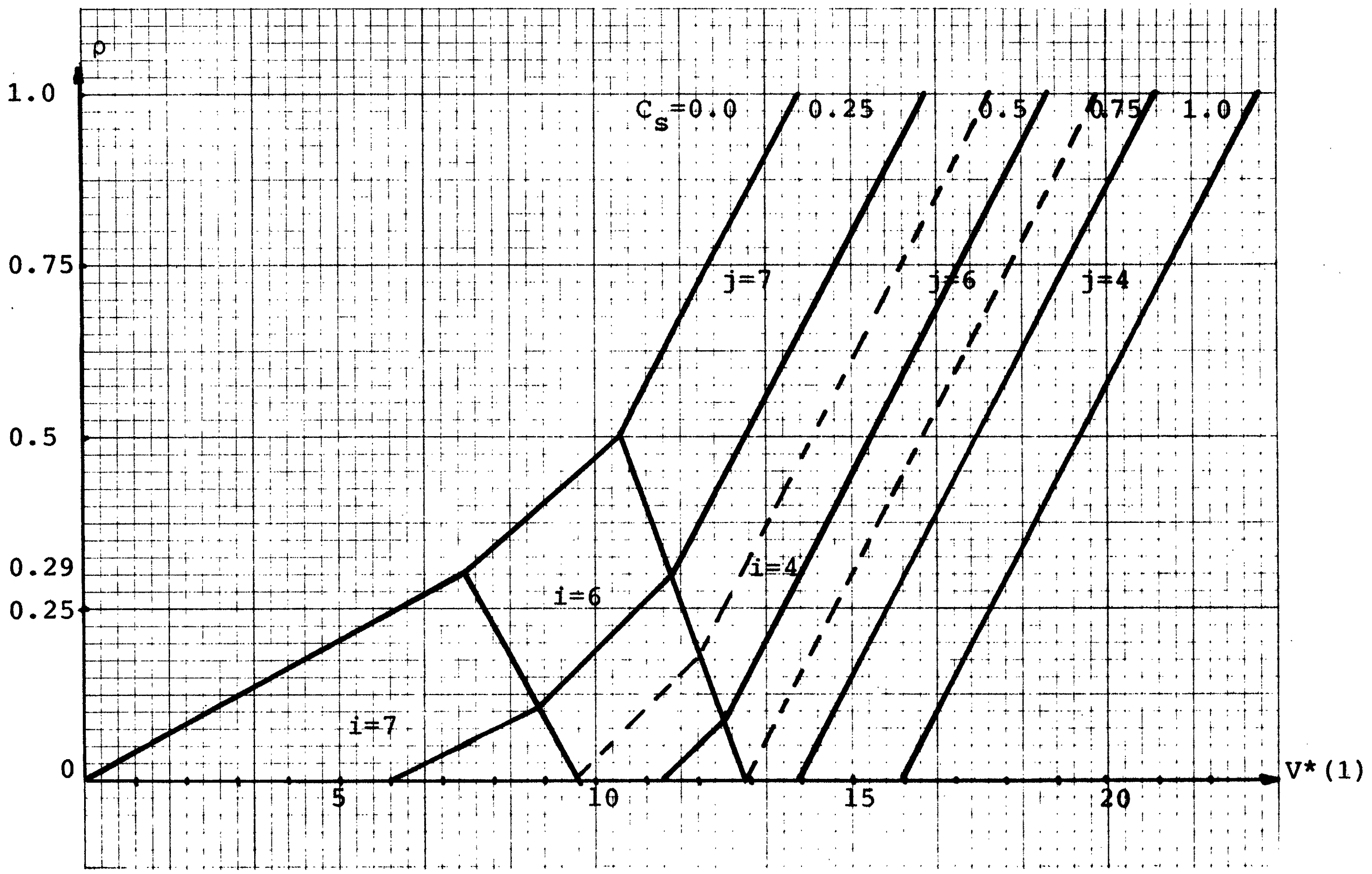


Fig. V.8b. Curves of the total minimum cost vs. ρ with C_s as a parameter. i represents trapping state and j optimum initial state.

$\theta_1(t) = \theta_1 = 0.5$

$\theta_2(t) = \theta_2 - 0.4$

$C_{1,2} = C_{2,1} = CT - 1$

$C_1 = C_2 = C_s - 0.5$

$P \equiv 0.25$

$C_{01} = 50 \quad C_{02} = 51$

$P_f - 0.01 \quad P_r = 0.1$

The terminal costs are given by (4.5) with $C_{oi}=0 \ \forall i$,

State number	State vector	Terminal costs
0	0 0	0.0
1	0 1	1.0
2	0 2	0.0
3	1 0	0.9
4	1 1	1.0
5	1 2	0.5
6	2 0	0.0
7	2 1	0.5
8	2 2	0.0

The results of the optimization process for the first iterations are shown as follows:

where under "cost" we write the values of the costs-to-go ordered from state 0 in top to state 8 in bottom, after every iteration. Under "go to" we list the optimal decision, corresponding to every state. The stars mean that for these states no decisions are needed.

Continuing the iterations we could see that the decision "go to state 4" is in fact the steady state optimal solution for states 1, 3 and 4. This fact can also be confirmed by applying the Howard algorithm 20 to this problem.

Therefore, for this problem, with the specified parameters, the optimal steady state decision is to keep as many copies as possible, that is one copy at each computer.

If keeping now the other parameters fixed we reduce the probability of failure by a factor of 10, i.e. from 0.01 to 0.001, we will find that the optimal steady state decision is "go to state 3", that is keeping only one copy at computer one. The first iterations are shown in table V.6.

Time:	T=1		T=2		T=3		T=4	
State	COST	go to	COST	go to	COST	go to	COST	go to
0	50.887040	**	52.368071	**	53.723572	**	55.004053	**
1	2.046306	3	3.412692	4	4.684624	4	5.908048	4
2	50.534600	**	51.159993	**	51.940678	**	52.836590	**
3	1.885050	3	3.254310	4	4.544083	4	5.785912	4
4	1.985050	3	3.186653	4	4.377834	4	5.561686	4
5	1.034600	**	1.659993	**	2.440678	**	3.336590	**
6	51.544500	**	52.183768	**	52.976518	**	53.881401	**
7	1.044500	**	1.683768	**	2.476518	**	3.381401	**
8	0.000000	**	9.695989	**	17.678371	**	24.299264	**

Table V.5.

Time:	T=1		T=2		T=3		T=4	
State	COST	go to	COST	go to	COST	go to	COST	go to
0	50.898700	**	51.947242	**	52.994890	**	54.041756	**
1	2.060876	3	3.140043	3	4.202974	3	5.257465	3
2	50.539460	**	51.129394	**	51.773340	**	52.464214	**
3	1.898500	3	2.947042	3	3.994691	3	5.041556	3
4	1.998500	3	3.047042	3	4.094691	3	5.141556	3
5	1.039460	**	1.629394	**	2.273340	**	2.964214	**
6	51.549450	**	52.154496	**	52.815088	**	53.522452	**
7	1.049450	**	1.654496	**	2.315088	**	3.022452	**
8	0.000000	**	9.696989	**	17.669583	**	24.255270	**

Table V.6.

Therefore a reduction in the risk of node failures is reflected in a reduction in the number of copies. It is obvious that further reductions in the probability of failures will not have any influence at all in the optimum policy because we cannot have less than one copy in the system.

We have seen how changes in the failure probability may change the minimum number of copies to be kept in a system; but what will be the repercussion of changes in P_r, the probability of recovery? It can be easily seen that changes in P_r will not have as much influence in the optimal decision as changes in P_f for the case NC=2. The reason is that, for NC=2, P_r only appears in transition probabilities from states where no decisions are available. The influence of these state costs on the decisions from other states is reflected through smaller probabilities (assuming P_f relatively small) and in a relatively simmetric form. Only if P_f is near 1, the value P_r might have a certain importance.

An intuitive explanation of the fact that the value of P_f will have much more influence than the value of P_r can be given by observing that, no matter how fast the failed computers have been restored, if the system looses all the copies, then a high price has to be paid in order to bring a copy from outside.

For cases with NC>2 the situation is not so simple because then P_r may appear directly in transition probabilities with several decisions available. Nevertheless, we can say that in those cases, the larger the number of working computers in a particular state the smaller the influence of P_r, for $P_f<<1$.

Therefore P_r will affect more the decisions among states with large number of failed computers. The reason can be easily seen with one example. Consider a network with NC=5 computers. If we are in state 1 ⟶ (0 0 0 0 1) then the next important costs that will affect the optimum decision from 1 will be the costs of states with no failures in their components, and therefore no P_r in their probability expressions; all these costs are going to be multiplied by $(1-P_f)^5$, (remember $P_f<<1$). The next set of important states contributing to the decisions is the set of states with one failed computer. These terms will have a factor of $(1-P_f)^4P_f$. Continuing in this way we can see that the states that will reflect more the value of P_r, that is the states with a large number of 2's in their components, will be multiplied by very small weights; for instance a state with only one working computer will be affected by the term $P_f^4(1-P_f)\tilde{-}0$ if $P_f<<1$.

On the other hand if the present state is for instance, state (0 2 1 2 2), and we assume $P_r>>P_f$ then the important terms will be the terms affected by the weights $(1-P_f)^2(1-P_r)^3$, $(1-P_f(^2P_r(1-P_r)^2$ and $(1-P_f)P_f(1-P_r)^3$ and therefore P_r will increase its role in the optimal decision from these states compared to the former one.

An intuitive explanation to this fact can be given as follows:

If P_r has a high value, close to 1, then there are high probabilities of transitions from, say state (0 2 1 2 2) to states like (_ 0 _ 0 0) or (_ 2 _ 0 0); suppose now that computers 4 and 5 have high request rates and that transmission costs from computer 3 to computer 4 and 5 are much higher than transmission costs from computer 1 to the same computers, then a decision of writing a copy at computer 1 will probably be optimal. On the other hand, if P_r is close to 0 then the transition probabilities to the statesabove will be very small and other factors will influence the optimal decision.

Of course if P_f is near 1 then P_r will increase its role in all decisions. With the discussion above we have only confirmed that the model in fact reflects the physical intuition that as long as P_f remains very small, the probability of recovery is of no great importance in the system (remember the intuitive explanations given above). It is obvious that the previous discussion has been undertaken considering a fixed, not too small, cost (comparing to transmission costs) of bringing a copy from outside to the system, in the case of loosing all the copies. It is clear that these costs will play a similar but opposed role to the failure probabilities. The reason is trivial.

Let us consider now an example with 3 computers NC=3. The parameters of the network are the following:

PF		OPTIMAL DECISIONS "GO TO STATE "														
		0.01					0.001					0.0001				
Time		T-1	T-2	T-3	T-4	T-5	T-1	T-2	T-3	T-4	T-5	T-1	T-2	T-3	T-4	T-5
Present State																
0	0 0 0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
1	0 0 1	12	13	13	13	13	12	13	13	13	13	12	12	12	12	12
2	0 0 2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
3	0 1 0	12	13	13	13	13	12	13	13	13	13	12	12	12	12	12
4	0 1 1	12	13	13	13	13	12	13	13	13	13	12	12	12	12	12
5	0 1 2	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
6	0 2 0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
7	0 2 1	15	16	16	16	16	15	16	16	16	16	15	16	16	16	16
8	0 2 2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
9	1 0 0	12	13	13	13	13	12	13	13	13	13	12	12	12	12	12
10	1 0 1	12	13	13	13	13	12	13	13	13	13	12	12	12	12	12
11	1 0 2	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
12	1 1 0	12	12	12	13	13	12	12	12	12	12	12	12	12	12	12
13	1 1 1	12	12	12	13	13	12	12	12	12	12	12	12	12	12	12
14	1 1 2	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
15	1 2 0	15	16	16	16	16	15	16	16	16	16	15	16	16	16	16
16	1 2 1	15	16	16	16	16	15	16	16	16	16	15	16	16	16	16
17	1 2 2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
18	2 0 0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
19	2 0 1	21	22	22	22	22	21	22	22	22	22	21	22	22	22	22
20	2 0 2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
21	2 1 0	21	22	22	22	22	21	22	22	22	22	21	22	22	22	22
22	2 1 1	21	22	22	22	22	21	22	22	22	22	21	22	22	22	22
23	2 1 2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
24	2 2 0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
25	2 2 1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
26	2 2 2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table V.4.

$\theta_1(t) = \theta_1 = 0.8$

$\theta_2(t) = \theta_2 = 0.6$

$\theta_3(t) = \theta_3 = 0.4$

$C_{i,j} = CT = 1 \quad \forall i,j \;\; i,j \; \{1,2,3\}$

$C_1 = C_2 = C_3 = C_s = 0.25$

$\rho = 0.25$

$C_{01} = 1000 \quad C_{02} = 1001 \quad C_{03} = 1002$

$P_r = 0.1$

Now we write the optimal decisions corresponding to the first iteration for three values of P_f.

$P_f = 0.01, 0.001, 0.0001$

In all cases it can be checked that the decisions at time T-5 (and T-4) constitute already the steady state optimal policy.

In these three cases it can be very well seen how the decrease of the failure probability decreases the number of copies.

For P_f=0.01 the steady state policy is to keep always as many copies as possible, that is, all working computers will carry a copy. For $P_f = 0.001$ the optimal policy is to keep as many copies as possible except for state 12 and 13 where two copies are enough (remember the discussion on Table V.3). For $P_f = 0.0001$ the optimal policy says: if all computers are working keep only two copies (in 1 and 2), otherwise keep as many copies as possible.

The optimal decision from states 12 and 13 for the case

P_f=0.001 might appear surprising considering that from all other states the decision is to go to state 13. The reason for this apparently anomalous fact can be seen by observing that a direct movement to state 12 carries a greater risk of ending up in a state with only one copy, than if we try to move the system to state 13. Then the transition from 13 to 12 is automatic (see also the discussion on Table V.3).

Another fact that can be inferred from the table is that the optimal decisions always try to keep copies in the computers with highest request rate and smallest cost of bringing a copy from outside. Remember that in the present example

$\Theta_1>\Theta_2>\Theta_3$ and $C_{01}<C_{02}<C_{03}$

Looking at all the examples studied in this section we can see as a common point that in all cases the steady state optimal solution is reached after very few iterations. If we began the iterations with a set of different terminal costs this would not have been the case. This fact has been confirmed for several examples by evaluating eigenvalues and eigenvectors of the transission matrix. We expanded different terminal cost vectors in the matrix eigenvector base and observed that the vector Λ gives the quickest rate of convergence. Nonetheless, this fact could not be proven analytically. A good reason to believe that the chosen terminal costs are a good set of values for a good speed of convergence of the iterations is to think that with these terminal costs we let the system finish in a"natural and not forced" way because those terminal costs are similar to the immediate cost(except for certain corrections due to the C_{oi} costs that make, in general, the first decision in the iterations different from the others). Nevertheless, this fact will tell us that those terminal vectors are better than others but not necessarily giving rise to a quick convergence.

If this is indeed the case, we could say that because we are mainly interested in the steady state policy the election of the per unit time costs as terminal costs will reduce considerably the amount of computation needed to find the steady state optimal policy. Otherwise, we always have available the algorithm developed by Howard [20] that has been proven quite efficient for those kinds of problems. In order to implement this algorithm we could take advantage of the fixed zeroes position in the transition matrix to solve the system of equations that this system generates. Another point that can save certain amount of computation in the solution of the system of equations is the fact that all states with NC=1 "2"'s in the equivalent positions give rise to identical transition probabilities and hence to identical rows in the transition matrix. That is for NC=4.

$$\begin{array}{ll} 79 \equiv (2221) & \\ & \text{row 79 of } P(t,u) \equiv \text{row 78 of } P(t,u) \quad \forall t,u \\ 78 = (2220) & \end{array}$$

$$\begin{array}{ll} 77 \equiv (2212) & \\ & \text{row 77 of } P(t,u) \equiv \text{row 74 of } P(t,u) \quad \forall t,u \\ 74 = (2202) & \end{array}$$

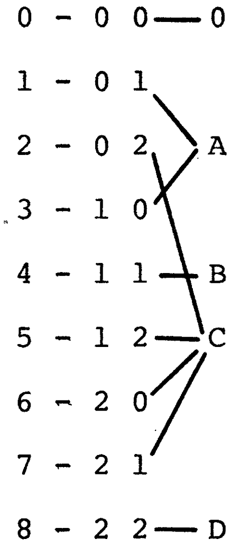
etc.

There have also been suggested in the literature e.g. [23], linear programming formulations to obtain an optimal policy using the principles of the policy improvement procedure of Howard's algorithm; these formulations solve the maximization problem involved in the policy improvement iteration of Howard's algorithm by means of linear programming calculations.

V.4 Completely Simmetric Network

In spite of the simplicity that the model provides to the analysis of a general network with any number of computers, there is one argument against it, as in fact is the case with many finite state formulations. The problem is the exponential growth of the number of states with the number of computers in the network grows. There is not much that can be done in order to avoid this growth but to try to find suboptimal solutions. One way in which these suboptimal schemes can be found is to assume that all transmission costs are equal, all rates are equal, all storage costs are equal etc. In such a situation we can see that all nodes in the network have the same role and there is no need to specify which node or nodes have a copy at a certain time but instead the state vector will contain only the information of the number of copies in the system at any particular time. Then the general Markov process, for this network, is mergeable, see 19 , in the sense that we can group together a certain number of states into a superstate and work with this superstate as if it was a simple state for the modified Markov process. In the case that for certain reasons, important difference in computer request rates for instance, we were interested in a further analysis, within some superstates, this can be done after the merged process was analyzed. Of course the results are not going to be optimal any longer but if we choose wisely the parameters good bounds to the optimal situation could be obtained.

For the case NC=2 we know that the states are 0-8, and can be grouped or merged as it is shown in (5.72)



(5.72)

It can be seen that this grouping verifies all the properties needed for a right merging, that is

$$\sum_{m \in S_1} P_{i,m} = \sum_{m \in S_1} P_{j,m} \quad \text{for } i,j \in S_k \quad \text{where } S_k,\ S_1 \in \{0,A,B,C,D\} \tag{5.73}$$

because now the indeces in the control variables are meaningless and $\theta_1 = \theta_2 = \theta$

The elements of the new transition matrix will be

$$\begin{array}{c|ccccc} & 0 & A & B & C & D \\ \hline 0 & 0 & (1-P_f)^2 & 0 & 2P_f(1-P_f) & P_f^2 \\ A & 0 & (1-P_f)^2(1-\alpha\Theta) & (1-P_f)^2\alpha\Theta & 2P_f(1-P_f) & P_f^2 \\ B & 0 & (1-P_f)^2\,\varepsilon & (1-P_f)^2(1-\varepsilon) & 2P_f(1-P_f) & P_f^2 \\ C & 0 & (1-P_f)P_r & 0 & (1-P_f)(1-P_r)+P_fP_r & P_f(1-P_r) \\ D & P_r^2 & 0 & 0 & 2P_r(1-P_r) & (1-P_r)^2 \end{array} \quad (5.74$$

where now α represents the fact of adding one copy and ε the fact of erasing a copy, no matter in which computer these actions will take place.

As an example let us see that if fact $P_{AA} = (1-P_f)^2(1-\alpha\Theta)$. We can write from the properties of the merging process

$$P_{AA}=P_{1,1}+P_{1,3}=P_{3,1}+P_{3,3} \qquad (5.75)$$

where

$$\begin{aligned} P_{1,1} &= (1-P_f)^2(1-\alpha_1\Theta) \\ P_{1,3} &= (1-P_f)^2\varepsilon_2\alpha_1\Theta \\ P_{3,1} &= (1-P_f)^2\varepsilon_1\alpha_2\Theta \\ P_{3,3} &= (1-P_f)^2(1-\alpha_2\Theta) \end{aligned} \qquad (5.76)$$

omitting the indeces, that we saw are now meaningless,

$$P_{A,A} = (1-P_f)^2(1-\alpha\Theta+\varepsilon\alpha\Theta) \qquad (5.77)$$

but because the decision of going to state 0 is not an admissible decision and, furthermore, we do not want to allow contradictory decisions (erasing and writing simultaneously) we have

$$\varepsilon=0$$

and then

$$P_{A,A} = (1-P_f)^2(1-\alpha\Theta) \qquad (5.78)$$

We can see that with this rearrangement we have reduced a 9 state system to a 5 state system.

For NC=3 the reduction is even more drastic. The states can be grouped as follows

New state	Grouped state	Representative of the new state	
0	0	(0 0 0)	
A	1,3,9	(1 0 0)	
B	4,10,12	(1 1 0)	(5.79)
C	13	(1 1 1)	
D	2,5,6,7,11,15,18,19,21	(0 0 2)and(1 0 2)	
E	14,16,22	(1 1 2)	
F	8,17,20,23,24,25	(0 2 2)and(1 2 2)	
G	26	(2 2 2)	

So we had a reduction from 27 to 8 in the number of states. Now we had to define the new control variables.

α^1 add a new copy

α^2 add two new copies

ε^1 erase one copy (5.80)

ε^2 erase two copies

The process can be easily generalized.

CONCLUSIONS AND OPEN QUESTIONS

Throughout the chapters of this thesis we have developed a new model to handle the problem of optimal dynamic file allocation. The model had to be general enough to allow the study of problems such as: dynamic allocation with possible computer failure and optimal allocation when we have restrictions in the state space. The restrictions may take the form of a maximum number of copies allowed in the system at any instant of time or not allowing copies of the file simultaneously in two or more given computers. The use of two types of control variables, one for adding new copies to the system (α), and the other for erasing copies (ε), made easier the task.

First we stated the working hypothesis (sufficiently high link capacities, sufficient memory sizes, stochastic independence in the requesting process from different computers, etc.) that could allow us to work with each file separately and to model the system as a Markov process.

Having characterized the evolution of the system under a Markov process and being interested in finding the optimal dynamic file allocation such that the total cost were minimized we found in the stochastic dynamic programming an excellent tool to solve the problem.

We defined the state of the system as a vector with a number of components equal to the number of computers in the network. In that way each component of the vector would characterize the particular situation of each computer. We would have a 0 in the ith location if computer i were in working condition but with no copy stored in its memory; a 1 if computer i had a copy in its memory and a 2 if computer i were out of work (if failures are included in the model). With this representation we could think in the state of the system as being a base 3 reproduction of a certain decimal number (or base 2 reproduction when the probability of failure is considered equal to zero). Therefore we could identify the states with this number.

We showed that the states and control vectors exhibited some properties that allowed us to write mechanically the transition tableau. This transition tableau has proved of great utility in writing the recursive equations generated by the application of dynamic programming. In fact we found some rules that made it possible to construct algorithmic flow-charts to compute the transition probabilities in a very efficient way. Perhaps one of the next important points related with this algorithm is its property of being totally general as far as the number of computers is concerned.

We have also seen that updating traffic generated at some or all of the nodes can be easily incorporated in the analysis. We have given flow-charts showing how all these terms can be calculated in the same way as the per unit time costs.

One of the reasons why the flow-charts were found to be quite efficient is because they compute only the nonzero elements of every matrix and vector. This is an important fact if we consider that, for instance, in the transition matrix only approximately 30% of the components are nonzero. A flow-chart for the whole optimization process was also presented.

After the complete introduction of the model we applied it to several examples. First we consider the case of time varying rates with no failures and no updating traffic in the network. We studied for this case how the state dynamics changed as the storage cost was increased. The analysis confirmed the intuitive point that the maximum number of copies needed in the network decreased as the storage cost increased. We also compared, for this case, the dynamic analysis with some static analysis. We plotted the curves of total cost-versus storage cost for the static and dynamic analyses for two examples with different rates but with the same average rates over the period of operation. It was found that the curves for the dynamic analysis were very close. A third example with higher rates were also plotted showing higher costs.

Later on, the case of constant rates with updating traffic and no failure was studied in great detail. It was shown that for this case the Markov process fitted into the special class of Markov chains with a trapping state. This fact was used to derive a certain number of properties. One of the properties is that these processes with a terminal cost vector equal to the immediate cost vector do not present any transient in their decision policy. Furthermore, the minimum expected cost increases picewise linearly with time to go. One important outcome of these two results was to find some expressions relating the trapping state and the optimal initial state with the storage cost and the updating ratio. In that way we could study, without actually implementing the dynamic programming algorithm, how the optimal allocation changed as we vary either the storage cost or the updating ratio or both. Curves were also given to show the evolution of the total cost versus storage cost taking the updating ratio as a parameter, and the total cost versus updating ratio taking the storage cost as a parameter. It was shown that due to above mentioned properties, those curves could be drawn without actually using the dynamic programming algorithm.

Finally the case of nonzero failure probability was studied. It was shown how an increment in the failure probability may increase the number of copies to be stored in the network at any time. It was also shown how, in general, variations in the probability of computer recovery do not have a significant effect on the optimal decisions if the failure probability is reasonably small. Perhaps one of the most important findings for this case, that could not be proved analytically though, was the fact that taking the per unit time cost as the vector terminal costs the process converges very quickly to the steady state decision. This is an important fact that could save a lot of computation and would avoid the need to use the Howard algorithm. Otherwise Howard's algorithm could be used efficiently while taking advantage of the fixed zero position in the transition matrix and the fact that some of its rows are identical.

As it was pointed out earlier, one of the difficulties that the finite state model rises is the fast increase in the number of states when the number of computers increases. To overtake this difficulty a suboptimal method based on a completely simmetric network which can be thought as an approximation to the actual network was suggested. This approximation provides a reduced Markov chain whose state are collections of the states of the original process. The reduction in the number of states for NC=3 was from 27 to 8.

Some points remain still to be studied related to dynamic file allocation. As mentioned before, we found some convergence properties that could not be proved analytically. Furthermore other suboptimal models can be of interest for the case of large networks. For example, some a priori calculated bound in the maximum and minimum number of copies could reduce considerably the number of states. But perhaps one of the most appealing topics to be pursued in this area is including the situation when the rates of request are not perfectly known in advance. The main goal then would be to try to generalize Segall's results 11 for this problem to a broader framework as the one presented here for the case of deterministic rates. With this approach, one could also investigate decentralized schemes for dynamic file allocation where the decisions at every time, whether to write or erase a copy, are done locally by each computer and all computers work in a team to minimize the overall cost [28].

APPENDIX A

Transition Probabilities for NC=2

We defined

$P_{ij}(t,u) = \text{Prob}\{\underline{Y}(++1) = j \;\; \underline{Y}(t) = i\}$

In particular

$P_{11}(t,u) = \text{Prob}\{\underline{Y}(t+1) = 1 \;\; \underline{Y}(t) = 1\} =$

$= \text{Prob}\{\underline{Y}(t+1)=1 \;\; \underline{Y}(t)=1, n_1(t)=1\}\text{Prob}\{n_1(t)=1 \;\; Y(t)=1\}$

$+ \text{Prob}\{\underline{Y}(t+1)=1 \;\; \underline{Y}(t)=1, n_1(t)=0\}\text{Prob}\{n_1(t)=0 \;\; Y(t)=1\}$

but

$\text{Prob}\{n_1(t)=1|Y(t)=1\}= \text{Prob}\{n_1(t)=1\}= \theta_1(t)$

$\text{Prob}\{n_1(t)=0|Y(t)=1\}= 1-\theta_1(t)$

$\text{Prob}\{\underline{Y}(t+1)=1|\underline{Y}(t)=1, n_1(t)=1\}=(1-\alpha_1(t))(1-\varepsilon_2(t))$

$\text{Prob}\{\underline{Y}(t+1)=1|\underline{Y}(t)=1, n_1(t)=0\}= 1$

therefore

$P_{11}(t,u)=(1-\alpha_1(t))(1-\varepsilon_2(t)) \;\; \theta_1(t) + 1-\theta_1(t) =$

$=(1-\alpha_1(t))\theta_1(t)-(1-\alpha_1(t))\varepsilon_2(t)\theta_1(t)+1-\theta_1(t)$

but $(1-\alpha_1(t))\varepsilon_2(t) = 0$ for any value of $\alpha_1(t)$ and $\varepsilon_2(t)$ in the control space then

$P_{11}(t,u) = 1-\alpha_1(t)\theta_1(t)$

In the same way

$P_{12}(t,u)=\text{Prob}\{\underline{Y}(t+1)=2|Y(t)=1\}=$

$=\text{Prob}\{\underline{Y}(t+1)=2|Y(t)=1, n_1(t)=1\}\text{Prob}\{n_1(t)=1\}+$

$+\text{Prob}\{\underline{Y}(t+1)=2|\underline{Y}(t)=1, n_1(t)=0\}\text{Prob}\{n_1(t)=0\}$

but $\text{Prob}\{\underline{Y}(t+1)=2|\underline{Y}(t)=1, n_1(t)=0\}= 0$

therefore

$P_{12}(t,u) = \alpha_1(t)\ \varepsilon_2(t)\ \Theta_1(t)$

with the same procedure

$P_{13}(t,u) = \alpha_1(t)(1-\varepsilon_2(t))\ \Theta_1(t)$

$P_{21}(t,u) = \varepsilon_1(t)\ \alpha_2(t)\ \Theta_2(t)$

$P_{22}(t,u) = (1-\alpha_2(t))(1-\varepsilon_1(t))\ \Theta_2(t) + 1-\Theta_2(t)$

$= 1-\alpha_2(t)\Theta_2(t)$ because $\varepsilon_1(t)(1-\alpha_2(t)) = 0$

$P_{23}(t,u) = \alpha_2(t)(1-\varepsilon_1(t))\ \Theta_2(t)$

$P_{31}(t,u) = \varepsilon_1(t)(1-\varepsilon_2(t)) = \varepsilon_1(t)$ because $\varepsilon_1(t)\varepsilon_2(t) = 0$

$P_{32}(t,u) = \varepsilon_2(t)(1-\varepsilon_1(t)) = \varepsilon_2(t)$

$P_{33}(t,u) = (1-\varepsilon_1(t))\ (1-\varepsilon_2(t)) = 1-\varepsilon_1(t)-\varepsilon_2(t)$

APPENDIX B

Backward Equations for a Network with Two Computers and the Restriction of Only One Copy in the System.

We saw in section III-3 that the backward equations for the case NC=2 without restrictions in the state space were

$$\begin{bmatrix} V_1^*(t) \\ V_2^*(t) \\ V_3^*(t) \end{bmatrix} = \begin{bmatrix} C_2+C_{21}\theta_1(t) \\ C_1+C_{12}\theta_2(t) \\ C_1+C_2 \end{bmatrix} +$$

$$\begin{bmatrix} 1-\alpha_1^*(t)\theta_1(t) & \varepsilon_2^*(t)\alpha_1^*(t)\theta_1(t) & (1-\varepsilon_2^*(t))\alpha_1^*(t)\theta_1(t) \\ \varepsilon_1^*(t)\alpha_2^*(t)\theta_2(t) & 1-\alpha_2^*(t)\theta_2(t) & (1-\varepsilon_1^*(t))\alpha_2^*(t)\theta_2(t) \\ \varepsilon_1^*(t) & \varepsilon_1^*(t) & 1-\varepsilon_1^*(t)-\varepsilon_2^*(t) \end{bmatrix} \begin{bmatrix} V_1^*(t+1) \\ V_2^*(t+1) \\ V_3^*(t+1) \end{bmatrix}$$

If we restrict the system to have only one copy at any instant of time then state 3 is not allowed and we have to do two changes according to section III-5.

1) we eliminate the last row of every matrix in the equation
2) we add the probability of going to state 3 to the probability of remaining in the previous state.

Doing that the transition state becomes

$$\begin{bmatrix} 1-\alpha_1\theta_1+(1-\varepsilon_2)\alpha_1\theta_1 & \varepsilon_2\alpha_1\theta_1 \\ \varepsilon_1\alpha_2\theta_2 & 1-\alpha_2\theta_2+(1-\varepsilon_1)\alpha_2\theta_2 \end{bmatrix} =$$

$$= \begin{bmatrix} 1-\varepsilon_2\alpha_1\theta_1 & \varepsilon_2\alpha_1\theta_1 \\ \varepsilon_1\alpha_2\theta_2 & 1-\varepsilon_1\alpha_2\theta_2 \end{bmatrix}$$

Realizing now (looking at the tableau of section II-4) that for any allowed transition $\alpha_1 = \varepsilon_2$ and $\varepsilon_1 = \alpha_2$ we can write

$$\varepsilon_2\alpha_1 = \alpha_1$$
$$\varepsilon_1\alpha_2 = \alpha_2$$

and we arrive to the logical and expected result that we do not need the erasure variables. With this simplification the recursive equation is

$$\begin{bmatrix} V_1^*(t) \\ V_2^*(t) \end{bmatrix} = \begin{bmatrix} C_2+C_{21}\theta_1(t) \\ C_1+C_{12}\theta_2(t) \end{bmatrix} + \begin{bmatrix} 1-\alpha_1(t)\theta_1(t) & \alpha_1(t)\theta_1(t) \\ \alpha_2(t)\theta_2(t) & 1-\alpha_2(t)\theta_2(t) \end{bmatrix} \begin{bmatrix} V_1^*(t+1) \\ V_2^*(t+1) \end{bmatrix}$$

that is the same equation of ref. 11 except for the switching of subscripts due to different state definitions.

APPENDIX C

FLOW-CHARTS AND SUBROUTINE FORTRAN LISTINGS

```
C     MAIN PROGRAM OF THE OPTIMIZATION PROCESS FOR A NETWORK WITH NO
C     FAILURES (FLOW-CHART FIG.III-4) AND CONSTANT PARAMETERS
      EXTERNAL DECI
      INTEGER DECI,TIEMPO,STATS(64),IB2(10)
      REAL THETA(10),V(64),UPT(64),MIN,C(64),VV(64)
      L=6
      K=5
      READ(K,1000) N,TIEMPO
      READ (5,1010) (THETA(J),J=1,N)
      M=(2**N)-1
      CT=1.
      RHO=0.
      DO 260 IS=1,5
      A=0.
      DO 240 IC=1,5
      CS=A
      NTEMPO=TIEMPO-1
      CALL CALCS (CS,THETA,N,M,V,CT)
      WRITE(L,1080) (KL,KL=1,N)
      WRITE(L,1090) KK,(THETA(J),J=1,N)
    6 CONTINUE
      WRITE(L,1060) A,RHO
      WRITE(L,1030) (I,I=1,M)
      WRITE(L,1040) (V(I),I=1,M)
      WRITE(L,1050) (I,I=1,M)
      A=A+0.25
      CALL UPTRAF(N,M,RHO,THETA,CT,UPT)
      CALL CALCS (CS,THETA,N,M,C,CT)
      DO 230 II=1,NTEMPO
      NTIME=TIEMPO-II
      DO 215 INSTA=1,M
      CALL ROWN  (N,M,INSTA,THETA,V,UPT,LAMBOP,SCOST)
      STATS(INSTA)=LAMBOP
      VV(INSTA)= SCOST+C(INSTA)
  215 CONTINUE
      DO 220 KK=1,M
```

```
 220 V(KK)=VV(KK)
     WRITE(L,1020)   NTIME,(STATS(I),I=1,M)
     WRITE(L,1070)  (V(I),I=1,M)
 230 CONTINUE
 240 CONTINUE
     RHO=RHO+0.25
 260 CONTINUE
 270 CONTINUE
1000 FORMAT(2I3)
1010 FORMAT(16F5.0)
1020 FORMAT(' ',I3,1X,15I2)
1030 FORMAT(40X,15I6)
1040 FORMAT(' TERMINAL COSTS',T41,15F6.2)
1050 FORMAT(' TIME   EVOLUTION OF STATES'/T6,15I2)
1060 FORMAT(' STORAGE COST=',F4.2,10X,' UPDATING RATIO =', F4.2)
1070 FORMAT('+',T41,15F6.2)
1080 FORMAT('1TIME     RATES',10I8)
1090 FORMAT(I3,11X,10F8.2)
     STOP
     END
```

```
C     THIS SUBROUTINE WRITE N IN BASE 2 WITH NCOM DIGITS
C     NB2(I) IS THE I-TH COMPONENT
       SUBROUTINE BITS(N,NCOM,NB2)
       IMPLICIT REAL*8(A-H,O-Z)
       DIMENSION NB2(10)
       ND=N
       I=1
     5 ICO=ND/2
       NB2(I)=ND-(2*ICO)
       IF (ICO.LT.2) GO TO 10
       I=I+1
       ND=ICO
       GO TO 5
    10 I=I+1
       NB2(I)=ICO
     7 IF(I.GE.NCOM) GO TO 8
       I=I+1
       NB2(I)=0
       GO TO 7
     8 IF(NCOM.LE.1) GO TO 16
       NCO=NCOM/2
       DO 15 I=1,NCO
       J=NCOM-I+1
       INTER=NB2(I)
       NB2(I)=NB2(J)
       NB2(J)=INTER
    15 CONTINUE
    16 CONTINUE
       RETURN
       END
```

```
C    ASEMBLER VERSION OF SUBROUTINE 'BITS'
BITS      START
          STM    14,12,12(13)
          BALR   BASE,0
          USING  *,BASE
BASE      EQU    10
ADDR      EQU    9
RONE      EQU    2
KUANS     EQU    3
N         EQU    4
AC        EQU    7
          L      RONE,=F'1'
          L      ADDR,0(0,1)
          L      N,0(0,ADDR)
          L      ADDR,4(0,1)
          L      KUANS,0(0,ADDR)
          L      ADDR,8(0,1)
LOOP      SR     5,5
          SRDL   N,1
          SRL    5,31
          LR     AC,KUANS
          SR     AC,RONE
          SLL    AC,2
          ST     5,0(AC,ADDR)
          BCT    KUANS,LOOP
          LM     14,12,12(13)
          BR     14
          END
```

```
C     THIS SUBROUTINE WRITE N IN BASE 3 WITH NC COMPONENTS
C     NB3(I) IS THE I-TH COMPONENT
C     NO= NUMBER OF 0'S IN NB3
C     N1= NUMBER OF 1'S IN NB3
C     N2= NUMBER OF 2'S IN NB3
      SUBROUTINE BASE3(NC,N,NB3,NO,N1,N2)
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER NB3(10)
      ND=N
      LEA =0
      I=1
      NO=0
      N1=0
      N2=0
    5 ICO=ND/3
      NB3(I)= ND-(3*ICO)
    9 IF(NB3(I)-1) 10,11,12
   10 NO=NO+1
      GO TO 13
   11 N1=N1+1
      GO TO 13
   12 N2=N2+1
   13 IF(LEA.EQ.1) GO TO 7
      I=I+1
      IF(ICO.LT.3) GO TO 6
      ND=ICO
      GO TO 5
    6 NB3(I)= ICO
      LEA=1
      GO TO 9
    7 IF(I.EQ.NC) GO TO 8
      I=I+1
      NB3(I)=0
      NO=NO+1
      GO TO 7
    8 NCO=NC/2
```

```
      DO 15 I=1,NCO
      J=NC-I+1
      INTER=NB3(I)
      NB3(I)=NB3(J)
      NB3(J)=INTER
   15 CONTINUE
      RETURN
      END
```

```
C     UPDATING TRAFFIC FOR A NETWORK WITH NO FAILURES
      SUBROUTINE UPTRAF (NC,M,RHO,THETA,CT,UPT)
      DIMENSION UPT(64),THETA(10),IB2(10)
      DO 20 I=1,M
      A=0.
      I1=0
      CALL BITS(I,NC,IB2)
      DO 5 K=1,NC
      IF (IB2(K).EQ.1) I1=I1+1
    5 CONTINUE
      DO 10 J=1,NC
      N1=I1
      IF(IB2(J).EQ.1) N1=N1-1
   10 A=A+RHO*THETA(J)*N1*CT
      UPT(I)=A
   20 CONTINUE
      RETURN
      END
```

```
C     THIS SUBROUTINE CORRESPONDS TO THE FLOW-CHART OF FIG III-3
C     INCLUDING UPDATING TRAFFIC
      SUBROUTINE ROWN  (NC,M,N,THETA,V,UPT,LAMBOP,SCOST)
      DIMENSION THETA(10),PI(64),V(64),UPT(64)
      INTEGER NB2(10),LA2(10),LB2(10),NAUX(10),JB2(10),DECI
      CALL BITS(N,NC,NB2)
      Q=10000000.
      DO 70 LAMB=1,M
      AB=0.
      CALL BITS(LAMB,NC,LA2)
      NI=0
      N1=0
      DO 10 I=1,NC
      LB2(I)=0
      IF(LA2(I).EQ.0) GO TO 10
      LB2(I)=1
      N1=N1+1
      IF(NB2(I).EQ.1) GO TO 10
    5 NI=NI+1
      NAUX(NI)=I
   10 CONTINUE
      IF(NI.EQ.0) GO TO 50
      IB=0
      IF(N1.EQ.NI) IB=1
      NF =2**NI
      DO 40 JJ=1,NF
      J=JJ-1
      CALL BITS(J,NI,JB2)
      A=1.
      DO 30 K=1,NI
      IA=NAUX(K)
      LB2(IA)=JB2(K)
      A=A*((1-JB2(K))*(1.-THETA(IA))+JB2(K)*THETA(IA))
   30 CONTINUE
      IF(IB.EQ.0) GO TO 35
      IF(J.NE.0) GO TO 35
      L=N
      GO TO 36
   35 L=DECI(2,LB2,NC)
   36 PI(L)=A
      AB=AB+PI(L)*(V(L)+UPT(L))
   40 CONTINUE
      GO TO 60
   50 PI(LAMB)=1.
      AB=V(LAMB)+UPT(LAMB)
   60 IF(AB.GE.Q) GO TO 70
      Q=AB
      LAOP=LAMB
   70 CONTINUE
      LAMBOP=LAOP
      SCOST=Q
      RETURN
      END
```

```
C     PER -UNIT-TIME COSTS FOR A NETWORK WITH NO FAILURES
      SUBROUTINE CALCS(CS,THETA,N,M,C,CT)
      DIMENSION THETA(10),C(64),IBIT(10)
      DO 80 I=1,M
      C(I)=0
      CALL BITS(I,N,IBIT)
      DO 30 J=1,N
      KK=IBIT(J)
      IF(KK) 20,20,10
   10 C(I)=C(I)+CS
      GO TO 30
   20 C(I)=C(I)+THETA(J)*CT
   30 CONTINUE
   80 CONTINUE
      RETURN
      END
```

```
C     MAIN PROGRAM OF THE OPTIMIZATION PROCESS FOR A NETWORK
C     WITH FAILURES (FLOW-CHART FIG.IV-6)
       EXTERNAL DECI
       INTEGER DECI
       DIMENSION THETA(10),RHO(10),CO(10),V(750),VV(15,750),VECL(750)
      C,DIF(15,750),R(250,250)
       INTEGER TIME, STOGO(15,750)
       READ(5,1000) NC,TIME,PF,PR
       READ(5,1010) (THETA(I),I=1,NC)
       M=3**NC
       READ(5,1010) (RHO(I), I=1,NC)
       READ(5,1010) (CO(J), J=1,NC)
       CT=1.
       A=0.5
       DO 260 IS=1,2
       CS=A
       KA=1
       CALL PUTCOS(NC,M,CS,CT,CO,THETA,KA,V)
       IF (KA.EQ.0) GO TO 6
       WRITE (6,1001) (CO(J),J=1,NC)
       WRITE(6,1002) PF ,PR
       WRITE (6,1080) (KL,KL=1,NC)
       WRITE(6,1090) TIME,(THETA(J), J=1,NC)
     6 CONTINUE
       WRITE (6,1060) A,RHO(1)
       WRITE (6,1070) (KL,KL=1,TIME)
       WRITE (6,1077)
       DO 7 I=1,M
       VV(1,I)=V(I)
       STOGO(1,I)= 88888
     7 CONTINUE
       A=A+0.5
       NTIME=TIME-1
       KA=0
       CALL PUTCOS(NC,M,CS,CT,CO,THETA,KA,VECL)
       CALL UPTRA(NC,M,THETA,RHO,CT,R)
      DO 230 II=1,NTIME
      III=II+1
      DO 215 INSTA=1,M
      N=INSTA-1
      CALL ROWBMA(NC,M,N,THETA,PF,PR,V,CO,RHO,CT,LAMBOP,SCOST)
      STOGO(III,INSTA)= LAMBOP
      VV(III,INSTA)=VECL(INSTA)+SCOST
  215 CONTINUE
      DO 216 K=1,M
      DIF(III,K)=VV(III,K)-V(K)
  216 V(K)=VV(III,K)
  230 CONTINUE
      DO 30 I=1,M
      II=I-1
   30 WRITE (6,1050) II,(VV(KL,I),STOGO(KL,I), KL=1,TIME)
      WRITE(6,1040)
      DO 40 I=1,M
      II=I-1
   40 WRITE(6,1045) II,(DIF(KL,I),KL=2,TIME)
  240 CONTINUE
      DO 250 LN=1,NC
  250 RHO(LN)=RHO(LN)+0.5
  260 CONTINUE
 1000 FORMAT (2I5,2F10.5)
 1001 FORMAT ('1CO(I)= ',10F8.2)
 1002 FORMAT (' PF=',F8.6,10X,'PR=',F8.6)
 1010 FORMAT (16F5.0)
 1040 FORMAT (' DIFFERENCES IN VALUE BETWEEN CONSECUTIVE STATES')
 1045 FORMAT (22X,I2,5(3X,F12.6,5X))
 1050 FORMAT ( ,I4,3X,6(F12.6,3X,I2,3X))
 1060 FORMAT (' STORAGE COST =',F4.2,10X,' UPDATING RATIO =',F4.2)
 1070 FORMAT (1X,'  STEP ',6(9X,I2,9X))
 1077 FORMAT (1X,' STATE ',6('    COST      GO TO '))
 1080 FORMAT (' TIME     RATES',10I8)
 1090 FORMAT (1X,I2,11X,10F8.2)
      STOP
      END
```

```
C     THIS SUBROUTINE CORRESPONDS TO THE FLOW-CHART OF FIG.IV-3
      SUBROUTINE PUTCOS(NC,M,CS,CT,CO,THETA,KA,VECL)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION THETA(10),VECL(250),CO(10),KB3(10)
      MM=M-1
      DO 50 KK=1,MM
      K=KK-1
      CALL BASE3(NC,K,KB3,KO,K1,K2)
      IF(K1.NE.0) GO TO 10
      B=1000000.
      DO 5 L=1,NC
      IF(KB3(L).NE.0) GO TO 5
      IF(CO(L).GT.B) GO TO 5
      B=CO(L)
    5 CONTINUE
      BB=0.
      IF(KA.EQ.0) BB=B
      VECL(KK)=BB
      GO TO 50
   10 IF((K1.EQ.1).AND.(KO.EQ.0)) GO TO 30
      A=0.
      DO 20 L=1,NC
      IF(KB3(L)-1) 11,12,20
   11 A=A+(THETA(L)*CT)
      GO TO 20
   12 A=A+CS
   20 CONTINUE
      VECL(KK)=A
      GO TO 50
   30 VECL(KK)=CS
   50 CONTINUE
      VECL(M)=0
      RETURN
      END
```

```
C     THIS SUBROUTINE OBTAIN THE N-TH ROW OF THE TRANSITION PROBABILITY
C     MATRIX OF NETWORK WITH NC COMPUTERS AND NO FAILURES WHEN THE
C     DECISION IS 'GO TO STATE LAMB'.
C     THE CORRESPONDING FLOW-CHART IS IN FIG III-3
C     PI(J) IS THE J-TH ELEMENTS OF THE ROW
C     IND IS AN OUTPUT INDEX SUCH THAT
C         IND= 1 IF SOME PI(J) = 1
C         IND= 0 OTHERWISE
      SUBROUTINE ROWPRO(NC,N,LAMB,THETA,PI,IND)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION THETA(10),PI(64)
      INTEGER NB2(10),LA2(10),LB2(10),NAUX(10),JB2(10),DECI
      CALL BITS(N,NC,NB2)
      CALL BITS(LAMB,NC,LA2)
      NI=0
      N1=0
      IND=0
      MP=2**NC-1
      DO 3 I=1,MP
    3 PI(I)=0.
      DO 10 I=1,NC
      LB2(I)=0
      IF(LA2(I).EQ.0) GO TO 10
      LB2(I)=1
      N1=N1+1
      IF(NB2(I).EQ.1) GO TO 10
    5 NI=NI+1
      NAUX(NI)=I
   10 CONTINUE
      IF(NI.EQ.0) GO TO 50
      IB=0
      IF(N1.EQ.NI) IB=1
      NF =2**NI
      DO 40 JJ=1,NF
      J=JJ-1
      CALL BITS(J,NI,JB2)
      A=1.
      DO 30 K=1,NI
      IA=NAUX(K)
      LB2(IA)=JB2(K)
      A=A*((1-JB2(K))*(1.-THETA(IA))+JB2(K)*THETA(IA))
   30 CONTINUE
      IF(IB.EQ.0)  GO TO 35
      IF(J.NE.0)  GO TO 35
      L=N
      GO TO 36
   35 L=DECI(2,LB2,NC)
   36 PI(L)=A
   40 CONTINUE
      GO TO 60
   50 PI(LAMB)=1.
      IND=1
   60 CONTINUE
      RETURN
      END
```

```
C     COMPUTE UPDATING TRAFFIC MATRIX FOR A NETWORK WITH FAILURES
      SUBROUTINE UPTRA (NC,M,THETA,RHO,CT,R)
C     COMPUTE THE UPDATING TRAFFIC MATRIX ACCORDING TO
C     FLOW-CHART OF FIG.IV-2
      DIMENSION THETA(1),RHO(1),R(250,250),IB3(10),IFB3(10)
      DO 5 J=1,M
      DO 5 JJ=1,M
    5 R(J,JJ)=0
      DO 50 I=1,M
      CALL BASE3(NC,I,IB3,IO,I1,I2)
      IF(I1.EQ.0) GO TO 50
      DO 40 IF=1,M
      CALL BASE3(NC,IF,IFB3,IFO,IF1,IF2)
      IF(IF1.EQ.0) GO TO 40
      IF(IF2.EQ.(NC-1)) GO TO 40
      DO 30 KX=1,NC
      IF(IB3(KX).EQ.2) GO TO 30
      NX=IF1
      IF(IFB3(KX).EQ.1) NX=IF1-1
      R(I,IF)=R(I,IF)+(RHO(KX)*THETA(KX)*NX*CT)
   30 CONTINUE
   40 CONTINUE
   50 CONTINUE
      RETURN
      END
```

```
C     THIS SUBROUTINE CORRESPONDS TO THE FLOW-CHART OF FIG.IV-5
       SUBROUTINE ROWBMA (NC,M,N,THETA,PF,PR,V,CO,R,LAMBOP,SCOST)
       IMPLICIT REAL*8(A-H,O-Z)
       DIMENSION THETA(10),NB3(10),PI(64)        ,JB2(10),P(250),V(250),
      XBB(250),CO(10),IB2(10),IW2(10),R(250,250)
       INTEGER VB2(10),LB2(10),LB3(10),NUVEC(10),MUVEC(10),DECI
      X,LAOP2(10),LAOP3(10),LA2(10)
       DO 5 I=1,M
       P(I)=0.
     5 CONTINUE
       LAMBOP=88888
       IF(N.EQ.0) GO TO 200
       MM=M-1
       MMM=M-2
       CALL BASE3 (NC,N,NB3,NO,N1,N2)
       IF (N2.NE.0) GO TO 210
       Q=10000000.
       MPRIM=(2**NC)-1
       NU=DECI(2,NB3,NC)
       DO 400 LAMB=1,MPRIM
       BB(LAMB)=0.
       CALL ROWPRO (NC,NU,LAMB,THETA,PI,IND)
       CALL BITS(LAMB,NC,LA2)
       DO 90 L=1,MM
       LL=L+1
       IF(L.NE.MM) GO TO 6
       P(M)=PF**NC
       GO TO 74
     6 CALL BASE3 (NC,L,LB3,LO,L1,L2)
       IF(L2.EQ.0) GO TO 10
       DO 7 LK=1,NC
       IF (((NB3(LK).EQ.1).AND.(LA2(LK).EQ. 1)).AND.(LB3(LK).EQ.0)) GO TO
      C 90
     7 CONTINUE
       IF(IND.EQ.0) GO TO 8
       DO 4 LK=1,NC
      IF((LA2(LK).EQ.0).AND.(LB3(LK).EQ.1))  GO TO 90
    4 CONTINUE
      A=1.
      GO TO 9
    8 IF(L1.GT.0)  GO TO 30
      II=2
      GO TO 40
   30 II=1
   40 LL2=(2**L2)
      A=0.
      DO 70 JJ=II,LL2
      J=JJ-1
      CALL BITS(J,L2,JB2)
      K=0
      DO 45 I=1,NC
      IF(LB3(I).NE.2)  GO TO 47
      K=K+1
      VB2(I)=JB2(K)
      GO TO 45
   47 VB2(I)=LB3(I)
   45 CONTINUE
      NV=DECI(2,VB2,NC)
      IF(NV.EQ.0)  GO TO 70
   60 A=A+PI(NV)
   70 CONTINUE
    9 P(LL)=(PF**L2)*((1.-PF)**(NC-L2))*A
      GO TO 89
   10 MU=DECI(2,LB3,NC)
      IF(PI(MU))  90,90,12
   12 P(LL)=((1.-PF)**NC)*PI(MU)
   89 CONTINUE
   74 BB(LAMB)=BB(LAMB)+P(LL)*(V(LL)+R(N,LL))
   90 CONTINUE
      IF(BB(LAMB).GT.0)  GO TO 400
      Q=BB(LAMB)
      LAOP=LAMB
```

```
400 CONTINUE
    CALL BITS(LAOP,NC,LAOP2)
    LAMBOP=DECI(3,LAOP2,NC)
    SCOST=Q
    GO TO 500
200 DO 201 K=1,NC
201 NB3(K)=0
    B=100000.
    DO 202 K=1,NC
    IF(CO(K).GT.B) GO TO 202
    B=CO(K)
    KK=K
202 CONTINUE
    J=KK
    MI=1
    NF=NC
    N2=0
    NP=NC-1
    GO TO 404
210 NF=NC-N2
    IF(N2.GE.(NC-1)) GO TO 280
    IF(N1.EQ.0) GO TO 282
    MPRIM=(2**NF)-1
    Q=100000.
    K=0
    DO 220 I=1,NC
    IF(NB3(I).EQ.2) GO TO 220
    K=K+1
    NUVEC(K)=NB3(I)
220 CONTINUE
    NU=DECI(2,NUVEC,NF)
    DO 450 LAMB=1,MPRIM
    CALL BITS(LAMB,NF,LA2)
    BB(LAMB)=0.
    CALL ROWPRO(NF,NU,LAMB,THETA,PI,IND)
    DO 290 L=1,MM
```

```
      LL=L+1
      CALL BASE3(NC,L,LB3,LO,L1,L2)
      KIND=0
      K=0
      NZ=0
      ND=0
      DO 225 I=1,NC
      IF(NB3(I)-1) 223,222,221
  221 IF(LB3(I)-1) 215,290,216
  215 NZ=NZ+1
      GO TO 225
  216 ND=ND+1
      GO TO 225
  222 K=K+1
      IF((LA2(K).EQ.1).AND.(LB3(I).EQ.0)) GO TO 290
      GO TO 224
  223 K=K+1
  224 IF(IND.EQ.0) GO TO 225
      IF ((LA2(K).EQ.0).AND.(LB3(I).EQ.1)) GO TO 290
      KIND=1
      A=1.
  225 CONTINUE
      IF((LO-NZ).EQ.NF) GO TO 290
  230 NW=L2-ND
      IF(NW.NE.0) GO TO 240
      K=0
      DO 233 I=1,NC
      IF(NB3(I).EQ.2) GO TO 233
      K=K+1
      MUVEC(K)=LB3(I)
  233 CONTINUE
      MU=DECI(2,MUVEC,NF)
      IF(PI(MU)) 289,289,234
  234 P(LL)=(PR**NZ)*((1.-PR)**ND)*((1.-PF)**NF)*PI(MU)
      GO TO 289
  240 IF(NW.NE.NF) GO TO 250
```

```
      P(LL)=(PR**NZ)*((1.-PR)**ND)*(PF**NF)
      GO TO 289
  250 IF(KIND.EQ.1) GO TO 270
      IF(N1.EQ.0) GO TO 255
      II=1
      GO TO 260
  255 II=2
  260 A=0.
      NNW=(2**NW)
      DO 265 JJ=II,NNW
      J=JJ-1
      CALL BITS(J,NW,JB2)
      K=0
      KK=0
      DO 266 I=1,NC
      IF(NB3(I).EQ.2) GO TO 266
      K=K+1
      IF(LB3(I).NE.2) GO TO 267
      KK=KK+1
      VB2(K)=JB2(KK)
      GO TO 266
  267 VB2(K)=LB3(I)
  266 CONTINUE
      NV=DECI(2,VB2,NF)
      IF(NV.EQ.0) GO TO 265
      A=A+PI(NV)
  265 CONTINUE
  270 P(LL)=(PR**NZ)*((1.-PR)**ND)*(PF**NW)*((1.-PF)**(NF-NW))*A
  289 CONTINUE
   77 BB(LAMB)=BB(LAMB)+P(LL)*(V(LL)+R(N,LL))
  290 CONTINUE
      IF(BB(LAMB).GT.Q) GO TO 450
      Q=BB(LAMB)
      LAOP=LAMB
  450 CONTINUE
      CALL BITS(LAOP,NF,LAOP2)
      LX=0
      DO 460 JX=1,NC
      IF(NB3(JX).EQ.2) GO TO 455
      LX=LX+1
      LAOP3(JX)=LAOP2(LX)
      GO TO 460
  455 LAOP3(JX)=2
  460 CONTINUE
      LAMBOP=DECI(3,LAOP3,NC)
      SCOST=Q
      GO TO 500
  282 B= 1000000.
      DO 284 I=1,NC
      IF(NB3(I).EQ.2)    GO TO 284
      IF(CO(I).GT.B) GO TO 284
      B= CO(I)
      JK=I
  284 CONTINUE
      J=JK
  285 MI=1
      NP=NC-1
  404 BA=0.
      LF=2**NP
      DO 300 II= 1,LF
      I=II-1
      CALL BITS(I,NP,IB2)
      DO 288 K=1,NP
      IF(IB2(K).EQ.0) GO TO 286
      IW2(K)=2
      GO TO 288
  286 IW2(K)=0
  288 CONTINUE
  292 DO 299 MM=MI,2
  293 DO 298 K=1,NC
      IF(K-J) 294,295,296
  294 LB2(K)=IW2(K)
```

```
      GO TO 298
  295 LB2(K)= MM
      GO TO 298
  296 KK=K-1
      LB2(K)=IW2(KK)
  298 CONTINUE
      NZ=0
      NS=0
      DO 120 K=1,NC
      IF(K.EQ.J) GO TO 120
      IF(NB3(K).NE.2) GO TO 110
      IF(LB2(K).NE.0) GO TO 120
      NZ=NZ+1
      GO TO 120
  110 IF(LB2(K).NE.2) GO TO 120
      NS=NS+1
  120 CONTINUE
      IF(MI.EQ.1) GO TO 121
      LL=DECI(3,LB2,NC)+1
      P(LL)=(PR**NZ)*((1.-PR)**(N2-NZ))
      GO TO 139
  121 C= (PR**NZ)*((1.-PR)**(N2-NZ))*(PF**NS)*((1.-PF)**(NF-NS-1))
      L=DECI(3,LB2,NC)
      LL=L+1
      IF(MM.EQ.1) GO TO 130
      P(LL)=C*PF
      GO TO 139
  130 P(LL)=C*(1.-PF)
  139 BA=BA+P(LL)*V(LL)
  140 CONTINUE
  299 CONTINUE
  300 CONTINUE
      SCOST=BA
      GO TO 500
  280 IF(N2.NE.(NC-1)) GO TO 320
      DO 310 K=1,NC
      IF(NB3(K).EQ.2) GO TO 310
      J=K
      GO TO 285
  310 CONTINUE
  320 MI=2
      J=NC+1
      NP=NC
      GO TO 404
  500 RETURN
      END
```

BIBLIOGRAPHY

[1].- Kleinrock, L. "Analytic and Simulation Methods in Computer Network Design". Spring Joint Computer Conference, 1970

[2].- McQuillan, J.M. and Walden, D.C. "Seminar on Computer Networks. Applied Math 254" Harvard University. Spring, 1975

[3].- Segall, A. "Notes on Data Communication Networks". Course 6.263 MIT. 1975.

[4].- Chu, W.W. "Advances in Computer Communication". Artech House, 1974.

[5].- Abranson, N. and Kuo, F. "Computer-communications Networks". Prentice-Hall, 1974.

[6].- Kleinrock, L. "Communication Nets". McGraw Hill, 1964.

[7].- Kleinrock, L. "Queueing Systems Vol. I". John Wiley 1974.

[8].- Chu, W.W. "Optimal File Allocation in a Computer Network". IEEE Trans. on Comp. October 1969.

[9].- Casey, R.G. "Allocation of Copies of a File in an Information Network". Spring Joint Computer Conference 1972.

[10].- Belokrinitskaya, L.B. et all. "Distribution of Functions Between Central Processor and Peripheral Computer". Automatika and Telemekanika, January 1972.

[11].- Segall, A. "Dynamic File Assignment in a Computer Network". IEEE Trans. on Automatic Control Vol. AC-21 April 1976.

[12].- Lew, A. "Optimal Resource Allocation and Scheduling among Parallel Processes". From "Parallel Processing" (Ed. T. Feng) Springer-Verlag. Berlin, 1975.

[13].- Bellman, R. "Dynamic Programming". Princeton, N. J. Princeton University Press, 1957.

[14].- Bellman, R. and Dreyfus, S.E., "Applied Dynamic Programming". Princeton, N.J. Princeton U.P. 1962.

[15].- Gluss, B. "An Elementary Introduction to Dynamic Programming". Allyn and Bacon 1972.

[16].- Athans, M. "Notes on Dynamic Programming" E.E. and C.S. Department MIT.

[17].- Howard, R.A. "Dynamic Probabilistic System. Vol. II" Wiley, 1971.

[18].- Parzen, E. "Stochastic Processes". Holden-Day, 1962.

[19].- Howard, R.A. "Dynamic Probabilistic System, Vol. I" Wiley, 1971.

[20].- Howard, R.A. "Dynamic Programming and Markov Processes". MIT Press, 1960.

[21].- DeLeve, G. "Generalized Markovian Decision Processes, Part I". Mathematical Centre Tracts, No. 3. Amsterdam 1964.

[22].- Mahmoud, S. and Riordon, J.S. .Optimal Allocation of Resources in Distributed Information Networks". ACM Trans. of Database Systems. Vol. 1 No. 1, March 1976.

[23].- Derman, C. "Finite State Markovian Decision Processes". Academic Press, N.Y. 1970.

[24].- Kleinrock, L. "Queuing Systems. Vol. II: Computer Applications". John Wiley, 1975.

[25].- Gorenstein, S. "Data Migration in a Distributed Data Base". RC4239 IBM Research. February 22, 1973.

[26].- Grossman, D.D. "Data Migration in an Information Network". RC 3696 IBM Research January 19, 1972.

[27].- Kaufmann, A. and Cruou, R. "Dynamic Programming" Academic Press. New York, 1967.

[28].- Segall, A. and Sandell, N.R. Jr. "Dynamic File Allocation in a Computer Network: Decentralized Control". In preparation for IEEE Trans. on Automatic Control.